NETWORK ALIGNMENT ON BIG NETWORKS

BY

SI ZHANG

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, 2021

Urbana, Illinois

Doctoral Committee:

Associate Professor Hanghang Tong, Chair
Professor Jiawei Han
Associate Professor Danai Koutra, University of Michigan
Assistant Professor Edgar Solomonik

ABSTRACT

In the age of big data, multiple networks naturally appear in a variety of domains, such as social network analysis, bioinformatics, finance, infrastructure and so on. Network alignment, which aims to find the node correspondences across different networks, can integrate multiple networks from different sources into a world-view network. By mining such a world-view network, one may gain considerable insights that are invisible if mining different networks separably. Networks as one common data type, share the well-known 4Vs characteristics of big data, including *variety, veracity, velocity and volume*, each of which brings unique challenges to the big network alignment task.

Specifically, the *variety* characteristic of big networks depicts the rich information associated with multiple networks. Many prior network alignment methods find the node correspondences merely based on network structures while inevitably ignoring the rich node and/or edge attributes of networks. In the meanwhile, conventional methods often assume the alignment consistency among the neighboring node pairs, which could be easily violated due to the disparity among various networks.

Despite the emergence of the sites and tools that enable to link entities, there still exist the bottlenecks of collecting the networked data, such as the privacy issues in social networks. Thus, real-world networks are often noisy and incomplete with missing edges. However, it still remains a daunting task on how to deal with the incompleteness and analyze the robustness of network alignment owing to the *veracity* characteristic.

The *velocity* of big networks indicates that real-world networks are often dynamically changing. The dynamics behind multiple networks may benefit network alignment from the temporal information of nodes and edges in addition to the static structural information of networks. Yet, how to design the dynamic alignment model still remains an open problem.

Given the sheer *volume* of large-scale networks but relatively limited computational resources, the at least quadratic complexity of many prior network alignment methods is not scalable especially when aligning networks with a large number of nodes and edges. In this way, the efficiency issue has become a fundamental challenge of big network alignment.

The theme of my Ph.D. research is to address the above challenges associated with the 4Vs characteristics and align big networks. Note that we consider volume as an overarching goal so we can align big networks efficiently. First (*for variety*), to leverage attribute information of networks, we develop a family of algorithms FINAL that optimize the alignment consistency in terms of network structures and attributes and achieve an up to 30% improvement

in terms of the alignment accuracy over the comparison methods without attributes. We also develop a novel alignment method that displace node representations to be more comparable through non-rigid point set registration. Moreover, to address network disparity issue, we design an encoder-decoder architecture NetTrans that learns network transformation functions in a hierarchical manner. Besides, we design a relational graph convolutional network based model with an adaptive negative sampling strategy to strike a balance between alignment consistency and disparity. This developed method named NeXtAlign achieves an at least 3% performance improvement over the best competitor. Second (*for veracity*), we hypothesize that network alignment and network completion mutually benefit each other and develop an effective algorithm based on multiplicative update that outperforms baseline methods on incomplete networks. In addition, we provide a robustness analysis of network alignment against structural noise. Last (*for velocity*), we design a representation learning model on dynamic network of networks which can leverage temporal information underlying networks and is applied for dynamic network alignment task.

ACKNOWLEDGMENTS

First and foremost I am extremely grateful to my advisor Dr. Hanghang Tong for his adequate guidance, invaluable advice, continuous support, and patience that cannot be underestimated during my Ph.D. study. Inspired by his sharp insights and broad visions, my first research work which has been mostly done during my master study was accepted by KDD'16. Up to now, my friends still joke on me that my debut is the pinnacle. Nourished by his tremendous help and advice, I learned a lot how to conduct research: from discovering research problems to figuring out novel solutions, from writing well-polished research papers to giving well-prepared presentations. Dr. Tong also provided me great opportunities to empower me from full potential including leading a 4-year research project, writing grant proposals, attending PI meetings, giving guest lectures in graduate-level courses and mentoring students. Besides all his constructive suggestions and unwavering guidance on my research, I have benefited a lot and will constantly benefit in the future from what he said - doctoral studies train one's research abilities and more importantly the endurance capability of facing obstacles. This together with all his encouragement helped me to get through the hard time, when the experiments did not go well, when my submissions got rejected. His optimism and his principle of 'Be Kind' that he introduced in CS 591 Ph.D. Orientation class, made my entire Ph.D. study much less stressful.

I would also like to say thank you to all the other thesis committee members, Dr. Jiawei Han, Dr. Danai Koutra and Dr. Edgar Solomonik for giving me valuable feedback and asking insightful questions during my thesis proposal and defense. Specifically, I am grateful to Dr. Han for sharing his deep and broad knowledge in the field of data mining and text mining in his CS 512 Data Mining Principles. I would like to thank Dr. Koutra for her great works on network alignment, which not only brought me into this research area but also inspired me a lot. I would like to express my sincere gratitude to Dr. Solomonik for all his constructive comments and suggestions on my research.

I have had a great journey of internship in Facebook AI, working on applied research problems. I would like to extend my sincere thanks to my mentor Long Jin and all the collaborators Yan Shang, Kai Wang, Yinglong Xia, Liang Xiong and Yunsong Guo. The diversity of the group has given me a great chance to learn from the experts in different domains and discover inspirational ideas. Their generous support and guidance helped me a lot to get used to the new industrial work environment, become familiar with the code base and accomplish industrial projects with nice outcomes.

As a member of Intelligent Data Engineering and Analytics Lab (iDEA) and iSchool Statistical Machine Learning And Artificial Intelligence Lab (iSAIL), I am grateful for the consistent support and encouragement from all other members: Liangyue Li, Chen Chen, Boxin Du, 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. In addition, I would like to thank all my friends at and outside of Arizona State University and University of Illinois at Urbana-Champaign, who are all imperative parts in this amazing journey.

Last but not least, I am deeply indebted to the most amazing parents, who have been extremely supportive during my graduate study in the United States. They have always been there ready to chat with me, provide encouragement and advice, celebrate my successes, help me recover fast after failures, and cheer me up. I would like to thank my wife Ms. Liya Chen for standing by me and her conjugial love.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 MOTIVATIONS

Multiple networks emerge in numerous domains ranging from social networks of various social platforms (e.g., Facebook, Twitter, etc.) and protein-protein interaction (PPI) networks of different species (e.g., yeast and human) to transaction networks at different financial institutes (e.g., PayPal and Zelle, etc.) and knowledge graphs constructed by multiple knowledge bases (e.g., DBPedia and FreeBase). These networks exhibit their inherent patterns and describe distinctive information which empowers numerous graph mining tasks on each single network. However, by merely mining each individual network separately, one may overlook the insights that are discoverable only by jointly mining multiple networks. In fact, networks are often inter-linked with each other at both macro level (e.g., graph similarities among graphs) and micro level (i.e., node level). Examples of being inter-linked at node level include social networks which share a set of common users, PPI networks with a set of overlapped proteins for certain functionalities and transaction networks with users that simultaneously involve in multiple financial institutes. From this perspective, it is often a key step to link nodes across different networks in many applications.

Network alignment aims to find the node correspondences across multi-sourced networks, which can be used to bridge networks at node level and integrate them into a world-view network. One example of social network alignment is illustrated in Figure 1.1 where the correspondence between two nodes across networks indicates these two nodes represent the same physical person. By finding the alignments among users, one can study how news and posts propagate within each social network and across different social networks [1]. Moreover, by aligning proteins across different PPI networks, knowledge can be transferred from well-studied species to less-studied species so that the evolutionary relationships among different species can be discovered [2, 3]. In addition, money launderers may conduct less suspicious transactions in each transaction network separately to camouflage themselves. With the help of network alignment, the merged transaction network from different financial institutions is more reliable to indicate fraud suspiciousness [4].

1.2 RESEARCH CHALLENGES

Despite the extraordinary importance, network alignment faces unique challenges brought from the characteristics of big networks (in a similar notion as big data), including the classic

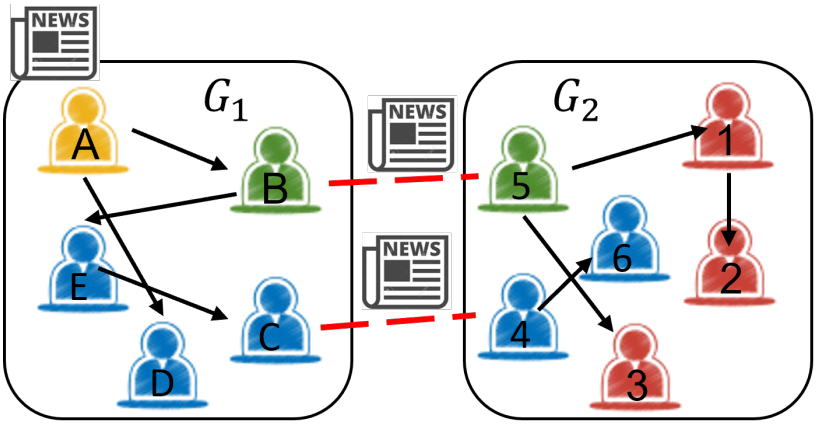


Figure 1.1: An illustration of network alignment between two social networks. Red dashed lines indicate the unique persons that participate in both social platforms.

variety, veracity, velocity and *volume*.

1.2.1 Variety Challenge

Despite the extensive research on network alignment, many of the alignment methods are solely based on the topology underlying networks [3, 5, 6]. However, the alignment achieved only by network structures might be sub-optimal or even incorrect, for example, when nodes are indistinguishable by topological information. In the era of big data, networks are often accompanied with rich contextual information, such as node and edge attributes, which could provide extra clues of node alignments. For example, user's profile (e.g., gender) may be used as categorical attributes that help remove the misleading alignment candidates. However, it is not clear how to assimilate node/edge attribute information into the alignment process and further mitigate both computational and analytical issues.

On the other hand, most of the prior approaches are either based on the typical consistency assumption that neighboring node pairs tend to have consistent alignment scores [3, 7], or based on the hypothesis that node embeddings of different networks lie in the same embedding space [8]. However, due to the different sources of networks, the disparity among networks may easily violate the consistency assumption of the classic optimization-based methods, and lead to the space disparity issue of the embedding-based methods. Scenarios that encounter such network disparity challenges include: (1) same user may behave dramatically differently in different social networks (e.g., being active in Facebook but quiet in Twitter due to his/her usage preferences), and (2) networks to be aligned capture different perspectives of users' social demands (e.g., career-oriented LinkedIn vs. friendship-oriented

Facebook). In this way, it is crucial to consider both the alignment consistency and disparity in the alignment algorithms.

1.2.2 Veracity Challenge

Prior network alignment methods predominantly assume that the network structure is perfectly known a priori. The veracity characteristic of big networks indicates that real-world networks are often incomplete (e.g., with missing nodes/edges and missing attributes, etc.) due to the limited accessibility in data collections. In particular, when collecting the relationships among users in social networks, the protection of user's privacy may hide some friendships of certain users such that the obtained social networks are incomplete. Directly aligning incomplete networks may result in the misleading node alignments. It is also worth mentioning that network completion, as a separate task of network alignment, may introduce noisy edges to the input networks and hence even hurt the alignment performance. In this way, how can we align input networks when one or both of them are incomplete (e.g., with missing entries in the corresponding adjacency matrices of the input networks)? In the meanwhile, veracity also means the uncertainty and noise underlying networks and even small changes in the network structure may significantly change the node alignments. Therefore, it is of great theoretical values to analyze the robustness and reliability of network alignment methods against the noise in network structures.

1.2.3 Velocity Challenge

Most, if not all, of the prior network alignment methods deal with static networks. That is, there exists no temporal information of networks incorporated in the alignment process. Nevertheless, many real-world networks such as social networks and transaction networks are intrinsically dynamic with nodes/edges changing over time. Such dynamics could provide more information in the additional temporal dimension to avoid misleading node alignments and thus further advance the accuracy of the node alignments. However, it remains a daunting task on how to leverage the temporal information in network alignment.

1.2.4 Volume Challenge

In the era of big data, one common challenge for most data mining tasks is the scalability issue due to the very large scale of real-world data. For example, Facebook has roughly 2.9 billions of monthly active users in 2021 and Twitter has about 209 millions of daily

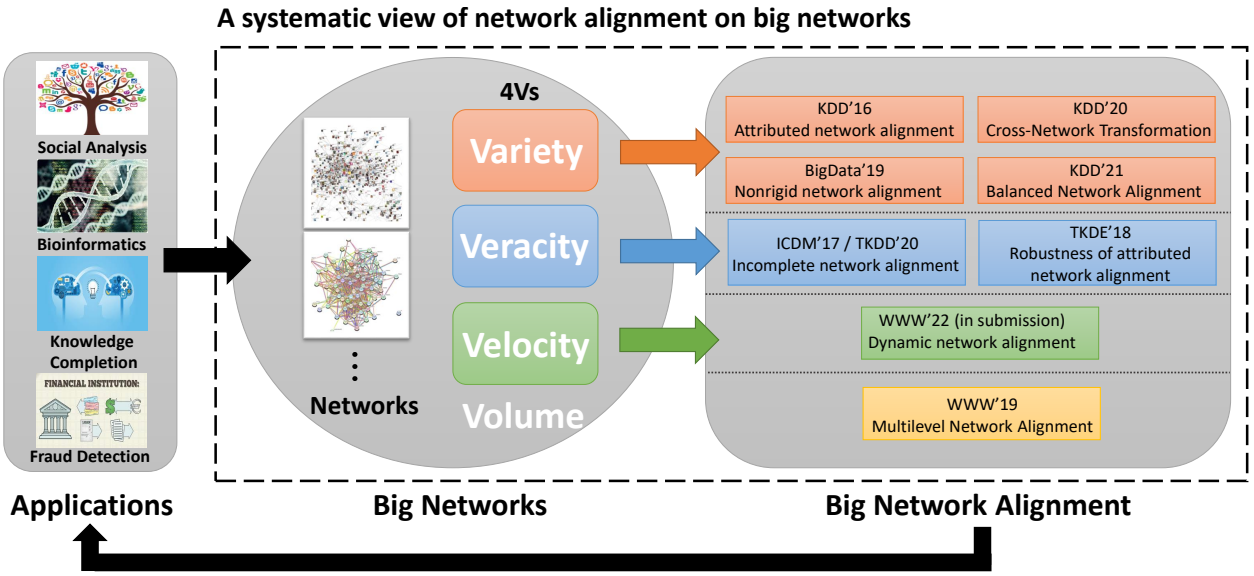


Figure 1.2: Theme of big network alignment with the scope of this research in the box.

active users. The substantial amount of users makes the computations of aligning the real-world social networks a huge challenge, especially considering that most network alignment methods have an at least quadratic computational complexity with respect to the number of nodes. To this end, the volume characteristic of big networks makes the complexity of alignment methods a fundamental challenge.

1.3 TASKS OVERVIEW

The overall goal of the thesis is to develop novel algorithms to align big networks that exist in numerous domains. Aligning big networks can facilitate the integration of multi-sourced networks into a world-view network which is further valuable to many high-impact applications. As shown in Figure 1.2, we systematically study the *big network alignment* problem while addressing the aforementioned challenges associated with the following three aspects (i.e., variety, veracity and velocity) with the volume as an overarching component. Note that to address the volume challenge, in addition to the developed methods that already have comparable computational complexity with prior methods, we also utilize the low-rank approximation [9, 10] and multi-resolution matrix factorization [11] to further scale up the algorithms.

Task 1 - Addressing Variety. As aforementioned, variety can be used to describe not only the various contextual information of nodes/edges, but also networks that serve different demands. To leverage node and edge attributes, we generalize the topological consistency

assumption by assimilating the attribute-based consistency and formulate it into an optimization problem. We develop a family of algorithms FINAL that achieve an up to 30% improvement in terms of the alignment accuracy [9, 12]. In addition, another task related to variety is to handle the network disparity, such as intrinsically different embedding space of different networks and different users' behavior patterns. It could mitigate the false positive alignment issues by addressing this task. Specifically, we design an embedding based method armed with non-rigid point set registration to handle the embedding space disparity challenge [13]. To resolve the network disparity issue, we alternatively address a more generic cross-network transformation problem and design an encoder-decoder architecture NetTrans that learns both the cross-network transformation functions and cross-network node associations. The NetTrans model achieves an up to 6.5% improvement in terms of Hits@30 compared with the comparison methods [14]. Moreover, we also want to balance the alignment consistency and disparity such that some false positive alignments can be avoided while not violating the overall alignment consistency. To this end, we develop a novel graph convolutional network based model to preserve alignment consistency and an adaptive negative sampling to incorporate alignment disparity [15].

Task 2 - Addressing Veracity. In this task, we first address the incomplete network alignment problem where the input networks have missing edges. Specifically, based on the hypothesis that network alignment and network completion can mutually benefit each other, we formulate it into an optimization problem and develop a multiplicative update to solve it with a *linear* time complexity [10, 16]. Empirically, this approach named iNeat achieves a better performance in both network alignment and network completion. Furthermore, given the nature that many networks are often noisy, we introduce a theoretical analysis on the robustness of the network alignment method [9], i.e., how robust the node alignments are against the noisy perturbations on network structures [12]. The key idea of this robustness analysis is the stability analysis of linear systems.

Task 3 - Addressing Velocity. Many real-world networks naturally change over time such as node and/or edge deletions/insertions. To exploit the temporal information underlying dynamic networks to provide extra clues how nodes should be aligned, we first reformulate the alignment problem into a node representation learning problem on network of networks. Then we develop a graph neural network model DraNoN armed with a gated recurrent unit and temporal attention mechanism to learn node representations on dynamic network of networks. As a result, the learned dynamic node representations are used to infer node alignments across dynamic networks. Experimental results demonstrate that DraNoN outperforms the prior static alignment methods.

1.4 ORGANIZATION

The remainder of the thesis is organized as follow. In Chapter 2, we will review the related literature on the variety, veracity, velocity and volume aspects of network alignment, as well as the applications of network alignment. In Chapter 3, we will introduce our works that resolve the alignment challenges associated with the variety characteristic. In Chapter 4, we will present our works on handling the veracity aspect of big network alignment. In Chapter 5, we will introduce how we deal with the velocity characteristic and leverage temporal information for dynamic network alignment. At last, we conclude and discuss the future research directions in Chapter 6.

CHAPTER 2: LITERATURE REVIEW

In this chapter, we review the related research on network alignment into the following parts: (1) variety in network alignment, (2) veracity in network alignment, (3) velocity in network alignment, (4) volume in network alignment, and (5) network alignment application.

2.1 BIG NETWORK ALIGNMENT - VARIETY

Most traditional network alignment methods are solely based on network structures [3, 5, 6, 17, 18, 19, 20]. These methods often explicitly or implicitly assume topological consistency in terms of alignment. That is, nodes are likely to be aligned if their corresponding neighbors are aligned. For example, Singh et al. proposed a classic alignment method that iteratively propagates the pairwise node similarity in a random walk fashion in the Kronecker product graph so that the similarities of neighboring node pairs are smooth [3]. Bayati et al. proposed a belief propagation based method to solve the constrained optimization problem that maximizes the number of neighboring alignment pairs according to both network structures and prior alignment matrix [5]. In addition, graph matching algorithms [21, 22, 23] are potentially applicable to compute node alignments. To be specific, Zhang et al. proposed to formulate multiple network alignment as a graph matching problem with transitivity constraints and solve for the permutation matrix with a sparsity relaxation [6]. More recently, many network embedding based methods have been proposed which aim to encode node structural information into low-dimensional node embeddings and use node embeddings to infer node alignments [8, 24, 25, 26, 27, 28]. For example, Liu et al. proposed to maximize the likelihood of observing the existing edges and known node alignment pairs across two networks calculated by node embeddings [8].

In the age of big data, real-world networks often have diversified node/edge attributes, which could facilitate network alignment. To leverage the rich contextual information associated with networks (e.g., node and edge attributes), many attributed network alignment methods have been proposed [7, 29, 30, 31, 32]. To incorporate attributes, Zhang et al. formulated both global structural consistency and local attribute consistency into an energy function [7]. In a closely related thread, most attributed graph matching methods can naturally use both graph structure and node/edge attributes by integrating them to an affinity matrix [33, 34, 35], but these methods do not scale well to large-scale networks. Furthermore, node attributes as well as the network structure can be used to jointly learn node embeddings which can subsequently infer node alignments. Heimann et al. proposed to

factorize a cross-network node similarity matrix that combines both structural and attribute similarities to obtain low-dimensional node embeddings, based on which node alignments can be inferred [29]. Zhang et al. proposed to learn both attribute embedding based on different views of node attributes and structural embedding by applying convolutional neural networks on new networks constructed upon the alignment rankings [30]. Note that the structure embeddings can capture the topological alignment consistency. With the advances of graph convolutional networks, node attributes can be naturally incorporated to learn node embeddings which can distinguish labelled alignment pairs and negative node pairs [31, 32].

On the other hand, networks from different sources exhibit disparity in structural patterns so that even aligned nodes may behave differently across networks. In this way, node embeddings of different networks may lie in totally different embedding space. To address this issue, Zhou et al. proposed to use multilayer perceptrons (MLP) to transform embedding space from one network to another network [25]. From another angle, Chu et al. proposed to disentangle node embedding vectors into network-specific node embeddings and network embedding shared by the nodes in the same network so node embeddings can capture network-specific structural patterns [27].

2.2 BIG NETWORK ALIGNMENT - VERACITY

To address the veracity challenges when aligning incomplete networks, the most straightforward way is to first complete the missing information in networks, followed by the typical network alignment process. There exist many network completion methods, which aim to reconstruct networks by inferring both unobserved nodes and edges [36, 37, 38, 39]. Specifically, under the assumption that the incomplete network follows Kronecker graph model, Kim and Leskovec proposed to use an EM algorithm to fit the observed network and estimate the missing part [36]. Without assuming specific graph model, Masrour et al. proposed to leverage node similarities to aid network completion with a theoretical guarantee [37]. Moreover, if only edges are missing, the classic matrix completion methods such as [40, 41, 42, 43] upon adjacency matrix can be also applied. In addition, link prediction can be also considered as a relevant topic to network completion which can be categorized into supervised methods [44, 45, 46], semi-supervised methods [47, 48] and unsupervised methods [49, 50]. Then, after network completion, networks are supposed to have a higher quality which further benefits the accuracy of network alignment. However, network completion itself may introduce some noisy edges which could even hurt the subsequent alignment process. In this way, a better way might be to conduct network completion and network alignment simultaneously. In particular, Du et al. proposed an iterative approach that alternatively

learns node embeddings based on the current network structures and infer node alignments based on which new edges can be imputed [51]. Phuc et al. proposed to simultaneously solve link prediction based on network embedding and the network alignment problem by optimal transport [52]. In a reverse direction, Zhang et al. used low-rank matrix completion technique to do link prediction across multiple networks that are already aligned [53].

In the meanwhile, real-world networks are often uncertain, noisy and vulnerable to the attacks, which makes it imperative to improve the robustness of the network alignment approaches. In terms of the uncertainty of networks, Zhou et al. proposed to learn node embeddings as Gaussian distributions rather than the typical point vectors in the embedding space, followed by an adversarial learning to infer node alignments [54]. Note that there also exist several works on single network embedding that captures the uncertainty by Gaussian embedding [55, 56]. If noisy networks are provided, directly conducting alignment algorithms may lead to misleading node alignments. One possible solution is to first reduce the structural noise behind each of the networks [57, 58], followed by any standard alignment methods. But this thread has not been systematically studied yet. Moreover, the vulnerability of networks to the adversarial attacks inspires two thrusts of research in the task of network alignment. The first thrust is to conduct adversarial attacks on networks to maximize the misalignment rate. Zhou et al. proposed to use the derivative over edges as the influence function upon Sylvester equation, based on which network structures are manipulated to degrade the alignment performance [59]. Zhang et al. proposed a kernel density estimation approach and a meta-learning based projected gradient descent method to perturb network structures to deceive the alignment model [60]. Another thrust is to improve the robustness of alignment methods against adversarial attacks. Specifically, Zhou et al. proposed an adversarial perturbation elimination model by combining Dirac delta approximation and adversarial perturbation elimination [61]. Ren at al. proposed a simplex detection technique to tackle the inter-graph dispersion attacks [62]. Other network alignment methods that are built upon adversarial learning include [63, 64, 65].

2.3 BIG NETWORK ALIGNMENT - VELOCITY

Despite the fact that many networks are intrinsically dynamic (e.g., social networks), most of the prior methods consider networks to be static. To align dynamic networks, one solution is to naturally extend the prior static alignment methods by adding temporal consistency measures, such as dynamic conservation measure [66] upon the static counterpart [20] and graph-orbit transition measures [67] upon the static version [68]. These heuristic methods depend on the specific kinds of networks (e.g., PPI networks) and thus lack flexibility of

applying to other scenarios (e.g., social networks). Sun et al. proposed an embedding based method that feeds node sequences to a LSTM based autoencoder with global consistency, and then learns node embedding vectors such that the aligned nodes coincide in the projected subspace [69]. However, this method assumes the node embedding vectors of different networks can be linearly transformed which might be sub-optimal to capture the complex alignments. Thanks to the temporal information, additional confidence can be provided to infer more accurate node alignments. Note that other dynamic network embedding methods could be applicable to be extended for dynamic network alignment. The options of dynamic network embedding include [70, 71, 72, 73, 74, 75]. For example, Zhou et al. proposed to use triadic closure to capture the evolution patterns [70]. Du et al. proposed to quantify the dynamic changes and update the embedding of nodes that are most effected [71]. Pareja et al. proposed an evolving graph convolutional network (GCN) model such that the weight parameter matrix in static GCN is updated by either LSTM or GRU [73]. Sankar et al. proposed a self-attention based method to learn the attention weights for embedding aggregation across the temporal axis [75].

Another thread of dynamic alignment is to efficiently update node alignments when dynamic changes occur, rather than computing node alignment from scratch. Yan et al. proposed a local updating strategy that utilizes the parameters of the base static model and updates a small part affected nodes' embeddings with a little accuracy loss efficiently [76]. In a related topic on dynamic graph kernel tracking, Li et al. proposed to leverage the low-rank characteristic of dynamic changes to incrementally update the eigen-decomposition of the adjacency matrices such that the graph kernel scores can be updated efficiently [77].

2.4 BIG NETWORK ALIGNMENT - VOLUME

To address the scalability challenges of network alignment, many approximation algorithms have been proposed. For example, Kollias et al. proposed to use rank-r singular value decomposition (SVD) to approximate the prior alignment matrix in IsoRank [3], which reduces the time complexity of each update iteration from $O(mn)$ to $O(n^2r)$ where m, n denote the number of edges and nodes, respectively [78]. In addition, Nassar et al. identify the low-rank structure of the alignment matrix in EigenAlign [79], achieving a super-linear alignment algorithm [80]. Chen et al. proposed to leverage the community structures underlying real-world networks to reduce the time complexity to sub-quadratic [81]. As many graph mining methods may be viewed as solving a Sylvester equation [9, 82], it is beneficial to efficiently solve the Sylvester equation. Du et al. proposed to project the original linear system into a Kronecker Krylov subspace, which with additional optimization techniques

achieves a linear computational complexity [83]. Yasar et al. proposed to apply divide-and-conquer and partition nodes into several buckets so that nodes of different networks in the same bucket are compared to decide node alignments [84]. In addition, Qin et al. proposed a two-level network alignment approach with the coarse-level networks constructed by graph compression [85].

2.5 NETWORK ALIGNMENT APPLICATIONS

Network alignment has been widely applied to diversified applications, such as social analysis, bioinformatics, knowledge completion and fraud detection. Specifically, user identity linkage is a natural application that find identical users across multiple social networks. In addition to many of the alignment methods that have been reviewed, Zafarani et al. proposed to identify users by modeling user behavior patterns based on human limitations, exogenous and endogenous factors [86]. Liu et al. proposed a method to identify same users by behavior modeling, structure consistency modeling and learning by multi-objective optimization [87]. Kong et al. utilized user's social, spatial, temporal and text information to infer the identical users [88]. A more comprehensive review on this application can be referred to [89]. In addition, by aligning social networks, knowledge of users' preferences can be transferred so that one can better recommend friends and products. For example, Yan et al. proposed to leverage random walks to integrate different aspects of information in one social platform for cold-start friend recommendation in another platform [90]. Cross-site link prediction can be also used for friend recommendation, including [91, 92, 93]. Similarly, by finding the alignments among users, items (e.g., products, news, posts, etc.) can be better recommended to different users. Cao et al. proposed a probabilistic graphical model for joint user modeling over aligned sites [94]. Zhu et al. proposed a topic model across multiple aligned networks to understand users topic preferences [95]. Moreover, Zhan et al. proposed to extend traditional linear threshold to the partially aligned networks to select a set of users who can maximize the spread of information [1].

In bioinformatics, network alignment is often a more powerful approach than sequence alignment to identify functional orthologs. Network alignment methods on PPI networks include but are not limited to [3, 17, 20, 96]. In addition, by aligning proteins across PPI networks, knowledge can be transferred from one species to another to reveal the evolutionary relationships across species and reveal new knowledge about human aging [2].

In many real-world scenarios, fraudsters often camouflage themselves such that their behavior just look normal [4]. By integrating the patterns of multi-sourced networks, the lack of sufficient fraud features of nodes can be somewhat complemented. Sun et al. proposed to

transfer features from one network to another based on node alignments such that abnormal subgraph can be detected [97]. Bindu et al. proposed to detect suspicious patterns in a multilayered network with different layers representing different social platforms [98].

A related application is knowledge completion by aligning entities across different knowledge graphs. The idea of completing relations is that the missing relation between two entities in one knowledge graph can be imputed if the aligned entities have relations in the other knowledge graph. Similarly, missing entities can be also completed. In general, recent knowledge graph alignment methods can be categorized into the embedding based methods [99, 100] and graph convolutional networks based methods [101, 102].

CHAPTER 3: VARIETY IN BIG NETWORK ALIGNMENT

Real-world networks are often accompanied with rich node and/or edge attributes. In addition, networks from different sources often serve for different demands, and hence exhibit disparities. In this chapter, we introduce our works on addressing the challenges resulted from the variety characteristic, including (1) attributed network alignment to assimilate attributes [9, 12], (2) non-rigid network alignment to address the space disparity issue of node embeddings [13], and (3) cross-network transformation to deal with network disparity [14]. In addition, we also study how to strike a balance between alignment consistency and alignment disparity in [15].

3.1 ATTRIBUTED NETWORK ALIGNMENT

Networks in many areas such as finance and social analysis, are often collected from multiple sources, leading to numerous emerging high-impact applications. However, an immense amount of these applications often require the knowledge of the relationships across multiple networks. Network alignment, on the other hand, is a powerful tool to explore the node correspondence amongst different networks, and has become the very first step to many applications. Finding the virtual identical twins across social networks, for instance, enables to measure the node proximity at a finer granularity [103]. Moreover, identification of the same customers in different transaction networks contributes to more comprehensive understandings of transaction behaviors and therefore helps detect financial fraud [104].

Meanwhile, many real-world networks are accompanied by rich numerical and categorical attribute information, including node attributes (e.g., user demographic information) and/or edge attributes (e.g., interaction information between users). Therefore, the fusion of both the topology consistency and attribute consistency might be a good cure to tackle these limitations. Researchers have been studying how to leverage the node attribute information and/or network structure to identify the unique users across different social networks [7, 86, 88]. For example, [88] extracts some structural features of each node from the networks and trains a binary classifier using the structural features and other node attribute information, to identify the unique users across multiple networks. More recently, CosNet models both local consistency based on the node attributes and global consistency based on the network structures into an energy-based model to predict the anchor links [7]. These identity matching based methods only endorse the attributes on nodes and require some identified nodes in advance to train the model with network structures and node attributes.

On the other side, the traditional graph matching approaches can encode both node and edge attributes together with the adjacency matrices into an affinity matrix [33, 34]. The nodes one-to-one mapping can be obtained by solving a nonconvex quadratic maximization problem. However, these methods are not scalable to the large-scale networks.

Alternatively, in this work, we shift our attention to the network alignment method that can not only use the topological information of the networks, but can also take advantages of both node and edge attributes. Yet, it still remains to be a daunting task to align attributed networks due to the following three challenges. First (*Q1. Formulation*), it is not clear how to assimilate both the node and edge attribute information into the topology-based network alignment and formulate it as a single optimization problem. Second (*Q2. Algorithms*), the optimization problem behind the topology-based network alignment is often non-convex or even NP-hard (e.g., maximum common subgraph optimization problem [5, 105]). Introducing attributes into the alignment process could further perplex the corresponding optimization problem. Third (*Q3. Computations*), while the scalability of the alignment algorithms is much desirable, it remains unknown how to accelerate and scale up the algorithms by taking advantage of some intrinsic properties (e.g., low-rank) of real networks.

To address these challenges, in this study, we propose a family of effective and efficient algorithms to solve the attributed network alignment problem. The key idea behind our algorithms is to generalize the *topology consistency* to *alignment consistency* and leverage attribute information to guide the topology-based alignment process. The algorithms can handle multiple numerical/categorical attributes. However, the computational challenges lie in the matrix multiplication between sparse adjacency matrix and the alignment matrix. Thanks to the low-rank structure of many real-world networks, we further propose an approximation algorithm for speed-up.

In some applications, we might be interested in finding similar nodes across different networks (e.g., to find similar users on LinkedIn for a given user on Facebook). We define this problem as the on-query attributed network alignment problem and develop a *linear* approximation algorithm without solving the full alignment problem.

The main contributions are summarized as follows.

1. *Formulations.* We define the attributed network alignment problem and formulate it as a *convex* quadratic optimization problem. As a side product, our formulation helps reveal the quantitative relationships between the (attributed) network alignment problem and several other network mining problems.

2. *Algorithms and Analysis.* We propose a family of algorithms FINAL to efficiently

Table 3.1: Symbols and notations.

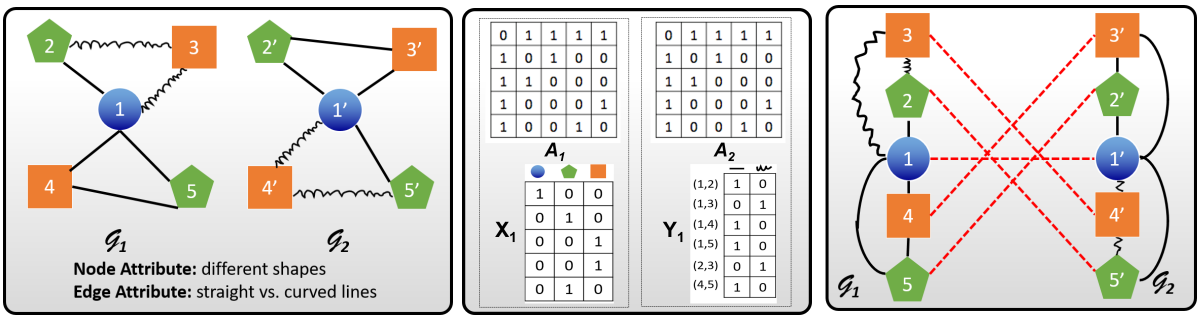
Symbols	Definition
$\mathcal{G} = \{\mathbf{A}, \mathbf{X}, \mathbf{Y}\}$	an attributed network
$\mathbf{A}$	the adjacency matrix of the network
$\mathbf{X}$	the node attribute matrix of the network
$\mathbf{Y}$	the edge attribute matrix of the network
n_1, n_2	# of nodes in $\mathcal{G}_1$ and $\mathcal{G}_2$
m_1, m_2	# of edges in $\mathcal{G}_1$ and $\mathcal{G}_2$
K, L	# of the node and edge attributes
a, b	node/edge indices of $\mathcal{G}_1$
x, y	node/edge indices of $\mathcal{G}_2$
v, w	node-pair indices of the vectorized alignment $\mathbf{s} = \text{vec}(\mathbf{S})$
$k(k'), l$	node/edge label indices
$\mathbf{I}, \mathbf{1}$	an identity matrix and a vector of 1s, respectively
$\mathbf{L}$	$n_1 \times n_2$ prior alignment preference matrix
$\mathbf{S}$	$n_1 \times n_2$ alignment matrix
r, p	reduced ranks
α	the parameter, $0 < \alpha < 1$
$\mathbf{a} = \text{vec}(\mathbf{A})$	vectorize a matrix $\mathbf{A}$ in column order
$\mathbf{Q} = \text{mat}(\mathbf{q}, n_1, n_2)$	reshape vector $\mathbf{q}$ into a $n_1 \times n_2$ matrix in column order
$\widetilde{\mathbf{A}}$	symmetrically normalize matrix $\mathbf{A}$
$\mathbf{D} = \text{diag}(\mathbf{d})$	diagonalize a vector
$\otimes$	Kronecker product
$\odot$	element-wise matrix product

solve the attributed network alignment problem. To speed up, we further develop an approximation algorithm to solve full alignment problem. We also develop a *linear* online algorithm for on-query alignment problem. We then analyze the optimality, convergence, complexity and stability.

3. *Evaluations.* We perform extensive experiments to validate the efficacy of the proposed algorithms.

3.1.1 Problem Definition

Table 3.1 summarizes the main symbols and notations used throughout this work. We use bold uppercase letters for matrices (e.g., $\mathbf{A}$), bold lowercase letters for vectors (e.g., $\mathbf{s}$), and lowercase letters (e.g., α) for scalars. We use $\mathbf{A}(i, j)$ to denote the entry at the intersection of the i-th row and j-th column of matrix $\mathbf{A}$, $\mathbf{A}(i, :)$ to denote the i-th row of $\mathbf{A}$ and $\mathbf{A}(:, j)$ to denote the j-th column of $\mathbf{A}$. We denote the transpose of a matrix by the superscript



(a) Input Attributed Networks. (b) Matrix Representation. (c) Alignment Output.

Figure 3.1: An illustrative example of the attributed network alignment problem. (a) Two input attributed networks. (b) The matrix representation for attributed networks, where two upper matrices represent the adjacency matrices, and the bottom matrices represent the node attribute and edge attribute matrices of $\mathcal{G}_1$ by using one hot encoding. (c) The desired alignment output (denoted by the red dashed lines).

prime (e.g., $\mathbf{A}'$ is the transpose of $\mathbf{A}$). We use˜on top to denote the symmetric normalization of a matrix (e.g., $\tilde{\mathbf{A}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, where $\mathbf{D}$ is the degree matrix of $\mathbf{A}$). The vectorization of a matrix (in the column order) is denoted by vec$(\cdot)$, and the resulting vector is denoted by the corresponding bold lowercase letter (e.g., $\mathbf{a} = \text{vec}(\mathbf{A})$).

We represent an attributed network by a triplet: $\mathcal{G} = \{\mathbf{A}, \mathbf{X}, \mathbf{Y}\}$, where (1) $\mathbf{A}$ is the adjacency matrix, and (2) $\mathbf{X}$ and $\mathbf{Y}$ are the node attribute and edge attribute matrices, respectively. The attributes of node-a corresponds to the vector of $\mathbf{X}(a,:)$, and $\mathbf{Y}_{(a,b)}$ describes the edge attribute vector of the edge between node-a and node-b. Note that for both node and edge categorical attribute values, they can be transformed into vectors by one hot encoding. Figure 3.1 presents an illustrative example. We can see from Figure 3.1(a), the set of nodes (2, 3, 4 and 5) from the first network share the exact same topology with another set of nodes (2′, 3′, 4′ and 5′) in the second network. The topology alone would be inadequate to differentiate these two sets. On the other hand, we can see that (1) 2, 2′, 5 and 5′ share the same node categorical attribute value; (2) 3, 3′, 4 and 4′ share the same node categorical attribute value; and (3) the two edges incident to 3 share the same edge categorical attribute value with those incident to 4′. These node/edge attributes could provide vital information to establish the accurate node-level alignment (i.e., 2 aligns to 5′, 5 aligns to 2′, etc.). This is exactly what this work aims to address. Formally, the attributed network alignment problem is defined as follows.

Problem 3.1. Attributed Network Alignment.

Given: (1) two undirected attributed networks $\mathcal{G}_1 = \{\mathbf{A}_1, \mathbf{X}_1, \mathbf{Y}_1\}$ and $\mathcal{G}_2 = \{\mathbf{A}_2, \mathbf{X}_2, \mathbf{Y}_2\}$ with n_1 and n_2 nodes respectively, (2 - optional) a prior alignment preference $\mathbf{L}$.

Output: the $n_1 \times n_2$ soft alignment/similarity matrix $\mathbf{S}$, where $\mathbf{S}(a, x)$ represents to what extent node-a in $\mathcal{G}_1$ is aligned with node-x in $\mathcal{G}_2$.

In the above definition, we have an optional input, to encode the prior knowledge of pairwise alignment preference $\mathbf{L}$, which is an $n_1 \times n_2$ matrix. An entry in $\mathbf{L}$ reflects our prior knowledge of the likelihood to align two corresponding nodes across the two input networks. For example, in the semi-supervised setting where some labeled node alignments (e.g., node-a aligned with node-x) are given, then we can construct the matrix $\mathbf{L}$ by setting $\mathbf{L}(a, x) = 1$. If no prior knowledge is provided, one can compute $\mathbf{L}$ by heuristics (e.g., degree similarities). Without loss of generality, we assume that $\mathbf{A}_1$ and $\mathbf{A}_2$ share a comparable size, i.e., $O(n_1) = O(n_2) = O(n)$ and $O(m_1) = O(m_2) = O(m)$. This will also help simplify our complexity analyses.

Notice that the alignment matrix $\mathbf{S}$ in Problem 3.1 is essentially a *cross-network node similarity* matrix, which naturally measures how similar nodes in $\mathcal{G}_1$ are with nodes in network $\mathcal{G}_2$. In some applications, we might be interested in finding a small number of similar nodes in one network w.r.t a query node from the other network. For instance, we might want to find the top-10 most similar LinkedIn users for a given Facebook user. We could first solve Problem 3.1 and then return the corresponding row or column in the alignment matrix $\mathbf{S}$, which might be computationally too costly as well as unnecessary. Having this in mind, we further define the on-query attributed network alignment problem as follows:

Problem 3.2. On-Query Attributed Alignment.

Given: (1) two undirected attributed networks $\mathcal{G}_1 = \{\mathbf{A}_1, \mathbf{X}_1, \mathbf{Y}_1\}$ and $\mathcal{G}_2 = \{\mathbf{A}_2, \mathbf{X}_2, \mathbf{Y}_2\}$, (2 -optional) a prior alignment preference $\mathbf{L}$, (3) a query node-a in $\mathcal{G}_1$.

Output: an $1 \times n_2$ vector $\mathbf{s}_a$ measuring similarities between the query node-a in $\mathcal{G}_1$ and all the nodes in $\mathcal{G}_2$ efficiently.

3.1.2 Topology Meets Attributes

We present our solutions for Problem 3.1. We start by formulating Problem 3.1 as a regularized optimization problem, and then develop effective algorithms to solve it, followed by some theoretic analysis.

FINAL: Optimization Formulation. The key idea behind our proposed formulation lies in the *alignment consistency* principle, which basically says that the alignments between two pairs of nodes across two input networks should be consistent if these two pairs of nodes

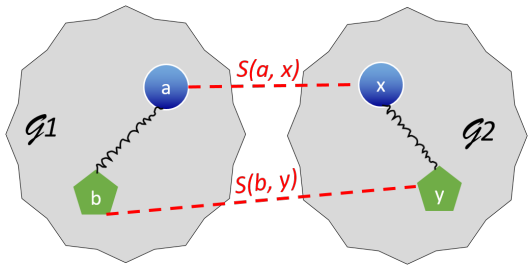


Figure 3.2: An illustration of alignment consistency.

themselves are similar/consistent with each other. Let us elaborate this using the following example. In Figure 3.2, we are given two pairs of nodes: (1) node-a in $\mathcal{G}_1$ and node-x in $\mathcal{G}_2$; and (2) node-b in $\mathcal{G}_1$ and node-y in $\mathcal{G}_2$. By the *alignment consistency* principle, we require the alignment between a and x, and that between b and y to be consistent (i.e., small $\|\mathbf{S}(a,x) - \mathbf{S}(b,y)\|$), if the following conditions hold:

C1 *Topology Consistency.* Node a and b are close neighbors in $\mathcal{G}_1$ (i.e., large $\mathbf{A}_1(a,b)$), and x, y are also close neighbors in $\mathcal{G}_2$ (i.e., large $\mathbf{A}_2(x,y)$);

C2 *Node Attribute Consistency.* Node a and x share the same or similar node attributes, and so do b and y;

C3 *Edge Attribute Consistency.* Edge (a,b) and (x,y) share the same or similar edge attributes.

The intuition behind the *alignment consistency* principle is as follows. If we already know that node-a is aligned to node-x (i.e., large $\mathbf{S}(a,x)$), then their close neighbors (e.g., b and y) with same or similar node attributes should have a high chance to be aligned with each other (i.e., large $\mathbf{S}(b,y)$), where we say that b and y are close neighbors of a and x respectively if they are connected by the same or similar edge attributes, with large edge weights (i.e., large $\mathbf{A}_1(a,b)$ and $\mathbf{A}_2(x,y)$). This naturally leads to the following objective function which we wish to minimize in terms of the alignment matrix $\mathbf{S}$:

$$J_1(\mathbf{S}) = \sum_{a,b,x,y} \left[\frac{\mathbf{S}(a,x)}{\sqrt{f(a,x)}} - \frac{\mathbf{S}(b,y)}{\sqrt{f(b,y)}} \right]^2 \underbrace{\mathbf{A}_1(a,b)\mathbf{A}_2(x,y)}_{\text{C1: Topology Consistency}} \times \underbrace{\Psi(a,x)\Psi(b,y)}_{\text{C2: Node Attribute Consistency}} \times \underbrace{\varphi((a,b),(x,y))}_{\text{C3: Edge Attribute Consistency}} \tag{3.1}$$

where (1) $a, b = 1, ..., n_1$, and $x, y = 1, ..., n_2$; (2) $\Psi(\cdot)$ is the function that measures the node attribute similarities between two nodes across networks; and (3) $\varphi(\cdot)$ measures the

edge attribute similarities between two edges in two networks. Besides, the function $f(\cdot)$ is a node-pair normalization function. For instance, we use the following function as $f(a, x)$

$$f(x, a) = \sum_{b,y} \mathbf{A}_1(a, b)\mathbf{A}_2(x, y)\Psi(a, x)\Psi(b, y)\varphi((a, b), (x, y)) \tag{3.2}$$

which measures how many (weighted) neighbor-pairs a and x have that (1) share the same or similar node attributes between themselves (e.g., b and y), and (2) connect to a and x via the same or similar edge attributes, respectively. Note that the functions $\Psi(\cdot)$ and $\varphi(\cdot)$ can be any existing similarity function. In this study, we use the cosine similarity to measure the similarity between node/edge attributes, i.e.,

$$\Psi(a, x) = \left(\frac{\mathbf{X}_1(a, :)}{\|\mathbf{X}_1(a, :)\|_2}\right)\left(\frac{\mathbf{X}_2(x, :)}{\|\mathbf{X}_2(x, :)\|_2}\right)' \tag{3.3}$$

$$\varphi((a, b), (x, y)) = \left(\frac{\mathbf{Y}_{1(a,b)}}{\|\mathbf{Y}_{1(a,b)}\|_2}\right)\left(\frac{\mathbf{Y}_{2(x,y)}}{\|\mathbf{Y}_{2(x,y)}\|_2}\right)' \tag{3.4}$$

where $\|\cdot\|_2$ is the vector L_2 norm, $\mathbf{X}_1, \mathbf{X}_2$ represent the node attribute matrices, and therefore $\mathbf{X}_1(a, :)$ is the feature vector of node-a. Besides, $\mathbf{Y}_{1(a,b)}, \mathbf{Y}_{2(x,y)}$ are the feature vectors of edge (a, b) in $\mathcal{G}_1$ and (x, y) in $\mathcal{G}_2$. To ease the computation, we denote $\mathbf{N}_1, \mathbf{N}_2$ as the normalized node attribute matrices where, for example, $\mathbf{N}_1(a, :) = \frac{\mathbf{X}_1(a,:)}{\|\mathbf{X}_1(a,:)\|_2}$. Next, we denote the edge feature vectors into a set of matrices. For edges in $\mathcal{G}_1$, we denote $\mathbf{Y}_1^l(a, b)$ as the l-th normalized attribute value of the edge (a, b) and $\mathbf{Y}_1$ is of same size as $\mathbf{A}_1$, i.e., $\mathbf{E}_1^l(a, b) = \frac{\mathbf{Y}_{1(a,b)(l)}}{\|\mathbf{Y}_{1(a,b)}\|_2}$. Similarly, we denote $\mathbf{E}_2^l(x, y)$ as the l-th normalized attribute value of edge (x, y). The cosine similarity functions $\Psi(\cdot)$ and $\varphi(\cdot)$ can be re-written as below.

$$\Psi(a, x) = \sum_{k=1}^{K} \mathbf{N}_1(a, k)\mathbf{N}_2(x, k) \tag{3.5}$$

$$\varphi((a, b), (x, y)) = \sum_{l=1}^{L} \mathbf{E}_1^l(a, b)\mathbf{E}_2^l(x, y) \tag{3.6}$$

Therefore, the objective function Eq. (3.1) can be written as

$$\begin{aligned} J_1(\mathbf{S}) = \sum_{a,b,x,y} [\frac{\mathbf{S}(a, x)}{\sqrt{f(a, x)}} - \frac{\mathbf{S}(b, y)}{\sqrt{f(b, y)}}]^2 \underbrace{\mathbf{A}_1(a, b)\mathbf{A}_2(x, y)}_{\text{C1: Topology Consistency}} \\ \times \underbrace{\sum_{k=1}^{K} \mathbf{N}_1(a, k)\mathbf{N}_2(x, k) \sum_{k'=1}^{K} \mathbf{N}_1(b, k')\mathbf{N}_2(y, k')}_{\text{C2: Node Attribute Consistency}} \times \underbrace{\sum_{l=1}^{L} \mathbf{E}_1^l(a, b)\mathbf{E}_2^l(x, y)}_{\text{C3: Edge Attribute Consistency}} \end{aligned} \tag{3.7}$$

where

$$f(a,x)=\sum_{b,y}\sum_{k,k'=1}^{K}\sum_{l=1}^{L}\mathbf{A}_1(a,b)\mathbf{A}_2(x,y)\mathbf{N}_1(a,k)\mathbf{N}_2(x,k)\mathbf{N}_1(b,k')\mathbf{N}_2(y,k')\mathbf{E}_1^l(a,b)\mathbf{E}_2^l(x,y) \tag{3.8}$$

Next, we present an equivalent matrix form of J_1, which is more convenient for the following algorithm description and the theoretical proof. By vectorizing the matrix $\mathbf{S}$ (i.e., $\mathbf{s}=\text{vec}(\mathbf{S})$), and with the notation of element-wise product and Kronecker product, Eq. (3.7) can be re-written as

$$\begin{aligned} J_1(\mathbf{s}) =& \sum_{v,w}\left[\frac{\mathbf{s}(v)}{\sqrt{\mathbf{D}(v,v)}}-\frac{\mathbf{s}(w)}{\sqrt{\mathbf{D}(w,w)}}\right]^2\mathbf{W}(v,w) \\ =& \mathbf{s}'(\mathbf{I}-\tilde{\mathbf{W}})\mathbf{s} \end{aligned} \tag{3.9}$$

where $v=n_1(x-1)+a$, $w=n_1(y-1)+b$, $\mathbf{W}=\mathbf{N}[\mathbf{E}\odot(\mathbf{A}_2\otimes\mathbf{A}_1)]\mathbf{N}$ and $\mathbf{N}=\text{diag}(\sum_{k=1}^{K}\mathbf{N}_2(:,k)\otimes\mathbf{N}_1(:,k))$, $\mathbf{E}=\sum_{l=1}^{L}\mathbf{E}_2^l\otimes\mathbf{E}_1^l$. $\tilde{\mathbf{W}}=\mathbf{D}^{-\frac{1}{2}}\mathbf{W}\mathbf{D}^{-\frac{1}{2}}$ is the symmetric normalized matrix of $\mathbf{W}$. The diagonal degree matrix $\mathbf{D}$ of $\mathbf{W}$ is directly derived from $f(a,x)$ and defined as

$$\mathbf{D}=\mathbf{N}\text{diag}(\sum_{k=1}^{K}\sum_{l=1}^{L}((\mathbf{E}_2^l\odot\mathbf{A}_2)\mathbf{N}_2(:,k))\otimes((\mathbf{E}_1^l\odot\mathbf{A}_1)\mathbf{N}_1(:,k))) \tag{3.10}$$

Note that some diagonal elements in $\mathbf{D}$ could be zero (e.g., $\mathbf{D}(v,v)=0$). For such elements, we define the corresponding $\mathbf{D}(v,v)^{-1/2}\triangleq 0$.

In some cases where we want to encode the prior alignment preference matrix $\mathbf{L}$ into the alignment result (e.g., in semi-supervised setting), we add a regularization term $\|\mathbf{s}-\mathbf{l}\|_2^2$ where $\mathbf{l}=\text{vec}(\mathbf{L})$. When no such prior information is given, we compute $\mathbf{l}$ by heuristics (e.g., degree similarities). From the optimization perspective, this additional regularization term would also help prevent the trivial solutions, such as a zero alignment matrix $\mathbf{S}$ or the alignment matrix $\mathbf{S}$ where $\mathbf{S}(a,x)=\sqrt{f(a,x)}$.

Putting everything together, our proposed optimization problem can be stated as follows.

$$\text{argmin}_{\mathbf{s}}\ J(\mathbf{s})=\alpha\mathbf{s}'(\mathbf{I}-\tilde{\mathbf{W}})\mathbf{s}+(1-\alpha)\parallel\mathbf{s}-\mathbf{l}\parallel_2^2 \tag{3.11}$$

where α is the regularization parameter.

We remark that when the node attributes are categorical attributes [9], we can one-hot encoding to transform categorical attributes into vector representations. To be specific, we briefly show that the indicator function on categorical attributes [9] is a special case of

cosine similarity. For example, consider countries as node attributes including {China, USA, Canada, Germany} and three nodes from USA, USA, Canada, respectively. The one-hot encoding of attribute value USA is represented as vector $(0, 1, 0, 0)$, and that of attribute value Canada is $(0, 0, 1, 0)$. Apparently, only when two nodes are both from USA, the cosine similarity of their node attributes is equal to 1 and otherwise 0, which is equivalent to the indicator function.

FINAL: Optimization Algorithms. The objective function in Eq. (3.11) is essentially a quadratic function w.r.t. $\mathbf{s}$. We seek to find its fixed-point solution by setting its derivative to be zero

$$\frac{\partial J(\mathbf{s})}{\partial \mathbf{s}} = 2(\mathbf{I} - \alpha\tilde{\mathbf{W}})\mathbf{s} + 2(\alpha - 1)\mathbf{l} = 0 \tag{3.12}$$

which leads to the following equation

$$\begin{aligned}\mathbf{s} &= \alpha\tilde{\mathbf{W}}\mathbf{s} + (1-\alpha)\mathbf{l} \\ &= \alpha\mathbf{D}^{-\frac{1}{2}}\mathbf{N}(\mathbf{E} \odot (\mathbf{A}_2 \otimes \mathbf{A}_1))\mathbf{N}\mathbf{D}^{-\frac{1}{2}}\mathbf{s} + (1-\alpha)\mathbf{l}\end{aligned} \tag{3.13}$$

We could directly develop an iterative algorithm based on Eq. (3.13). However, such an iterative procedure involves the Kronecker product between $\mathbf{A}_1$ and $\mathbf{A}_2$ whose time complexity is $O(m^2)$. Although the Kronecker product can be pre-computed, the $O(m^2)$ space complexity due to the memory cost and the $O(m^2)$ time complexity in each iteration due to the matrix-vector multiplication are still impractical for large networks.

In order to develop a more efficient algorithm, thanks to a key Kronecker product property (i.e., $\text{vec}(\mathbf{ABC}) = (\mathbf{C}' \otimes \mathbf{A})\text{vec}(\mathbf{B})$), we re-write Eq. (3.13) as

$$\mathbf{s} = \alpha\mathbf{D}^{-\frac{1}{2}}\mathbf{N}\text{vec}(\sum_{l=1}^{L}(\mathbf{E}_1^l \odot \mathbf{A}_1)\mathbf{Q}(\mathbf{E}_2^l \odot \mathbf{A}_2)) + (1-\alpha)\mathbf{l} \tag{3.14}$$

where $\mathbf{Q}$ is an $n_1 \times n_2$ matrix reshaped by $\mathbf{q} = \mathbf{N}\mathbf{D}^{-\frac{1}{2}}\mathbf{s}$ in column order, i.e., $\mathbf{Q} = \text{mat}(\mathbf{q}, n_1, n_2)$. We can show that Eq. (3.13) and Eq. (3.14) are equivalent with each other. The advantage of Eq. (3.14) is that it avoids the expensive Kronecker product, which leads to a more efficient iterative algorithm FINAL-NE (summarized in Algorithm 3.1).

Variants of FINAL-NE. Our proposed FINAL-NE algorithm assumes that the input networks have both node and edge attributes. It is worth pointing out that it also works when the node and/or the edge attribute information is missing.

First, when only node attributes are available, we can set all entries in the edge attribute matrices $\mathbf{E}^l$ to 1 where an edge indeed exists. The intuition is that we treat all the edges in

Algorithm 3.1: FINAL-NE: Attributed Network Alignment.

Input : (1) $\mathcal{G}_1 = \{\mathbf{A}_1, \mathbf{X}_1, \mathbf{Y}_1\}$ and $\mathcal{G}_2 = \{\mathbf{A}_2, \mathbf{X}_2, \mathbf{Y}_2\}$, (2) optional prior alignment matrix $\mathbf{L}$, (3) regularization parameter α, and (4) maximum iteration number $t_{\max}$.

Output: the $n_1 \times n_2$ alignment matrix $\mathbf{S}$ between $\mathcal{G}_1$ and $\mathcal{G}_2$.

Construct normalized node attribute matrices $\mathbf{N}_1$, $\mathbf{N}_2$;
Construct normalized edge attribute matrices $\mathbf{E}_1^l$, $\mathbf{E}_2^l$, $l = 1, \cdots, L$;
Compute the node attribute matrix $\mathbf{N}$ and degree matrix $\mathbf{D}$;
Initiate the alignment $\mathbf{s}$ as a uniform vector, and $t = 1$;
while $t \leq t_{max}$ **do**
    Compute vector $\mathbf{q} = \mathbf{N}\mathbf{D}^{-\frac{1}{2}}\mathbf{s}$;
    Reshape $\mathbf{q}$ as $\mathbf{Q} = \text{mat}(\mathbf{q}, n_1, n_2)$;
    Initiate an $n_1 \times n_2$ zero matrix $\mathbf{T}$;
    for $l = 1 \to L$ **do**
        Update $\mathbf{T} \leftarrow \mathbf{T} + (\mathbf{E}_1^l \odot \mathbf{A}_1)\mathbf{Q}(\mathbf{E}_2^l \odot \mathbf{A}_2)$;
    end
    Update $\mathbf{s} \leftarrow \alpha\mathbf{D}^{-\frac{1}{2}}\mathbf{N}\text{vec}(\mathbf{T}) + (1-\alpha)\mathbf{h}$;
    Set $t \leftarrow t + 1$;
end
Reshape $\mathbf{s}$ to $\mathbf{S} = \text{mat}(\mathbf{s}, n_1, n_2)$.

the networks to share a common edge attribute value. In this case, the fixed-point solution in Eq. (3.13) becomes

$$\mathbf{s} = \alpha\mathbf{D}_n^{-\frac{1}{2}}\mathbf{N}(\mathbf{A}_2 \otimes \mathbf{A}_1)\mathbf{N}\mathbf{D}_n^{-\frac{1}{2}}\mathbf{s} + (1-\alpha)\mathbf{l} \tag{3.15}$$

where $\mathbf{D}_n = \mathbf{N}\text{diag}(\sum_{k=1}^{K}(\mathbf{A}_2\mathbf{N}_2(:,k)) \otimes (\mathbf{A}_1\mathbf{N}_1(:,k)))$ denotes the degree matrix of $\mathbf{W}_n$. Similar to Eq. (3.14), we can use the vectorization operator to accelerate the computation. We refer to this variant as FINAL-N, and omit the detailed algorithm description.

Second, when only edge attributes are available, we treat all nodes to share one common node attribute value by setting $\mathbf{N}$ to be an identity matrix. In this case, the fixed-point solution in Eq. (3.13) becomes

$$\mathbf{s} = \alpha\mathbf{D}_e^{-\frac{1}{2}}(\mathbf{E} \odot (\mathbf{A}_2 \otimes \mathbf{A}_1))\mathbf{D}_e^{-\frac{1}{2}}\mathbf{s} + (1-\alpha)\mathbf{l} \tag{3.16}$$

where $\mathbf{D}_e = \text{diag}(\sum_{l=1}^{L}[(\mathbf{E}_2^l \odot \mathbf{A}_2)\mathbf{1}] \otimes [(\mathbf{E}_1^l \odot \mathbf{A}_1)\mathbf{1}])$. Again, we omit the detailed algorithm description, and refer to this variant as FINAL-E.

Finally, if neither the node attributes nor the edge attributes are available, Eq. (3.13)

degenerates to

$$\mathbf{s} = \alpha \mathbf{D}_u^{-\frac{1}{2}} (\mathbf{A}_2 \otimes \mathbf{A}_1) \mathbf{D}_u^{-\frac{1}{2}} \mathbf{s} + (1-\alpha)\mathbf{l} \tag{3.17}$$

where $\mathbf{D}_u = \mathbf{D}_2 \otimes \mathbf{D}_1$, $\mathbf{D}_1$ and $\mathbf{D}_2$ are the degree matrix of $\mathbf{A}_1$ and $\mathbf{A}_2$. This variant is referred to as FINAL-P.

Proofs and Analysis. We first analyze the *convergence*, *optimality* and *complexity* of our FINAL algorithms. Due to the space limit, we only present the results for the most general case (i.e., FINAL-NE). Then we analyze the relationships between FINAL and several classic graph mining problems.

We start with Lemma 3.1, which says the FINAL-NE algorithm converges to the global optimal solution of Eq. (3.9).

Lemma 3.1. Convergence and Optimality of FINAL-NE. Algorithm 3.1 converges to the closed-form global minimal solution of $J(\mathbf{s})$: $\mathbf{s} = (1-\alpha)(\mathbf{I} - \alpha\tilde{\mathbf{W}})^{-1}\mathbf{l}$.

Proof. Since $\tilde{\mathbf{W}}$ is similar to the stochastic matrix $\mathbf{W}\mathbf{D}^{-1} = \mathbf{D}^{\frac{1}{2}}\tilde{\mathbf{W}}\mathbf{D}^{-\frac{1}{2}}$, the eigenvalues of $\tilde{\mathbf{W}}$ are within $[-1, 1]$. Given $0 < \alpha < 1$, the eigenvalues of $\alpha\tilde{\mathbf{W}}$ are in $(-1, 1)$.

We denote the alignment vector $\mathbf{s}$ in the t-th iteration as $\mathbf{s}^{(t)}$. We have that

$$\mathbf{s}^{(t)} = \alpha^t \tilde{\mathbf{W}}^t \mathbf{l} + (1-\alpha) \sum_{i=0}^{t-1} \alpha^i \tilde{\mathbf{W}}^i \mathbf{l} \tag{3.18}$$

Since the eigenvalues of $\alpha\tilde{\mathbf{W}}$ are in $(-1, 1)$, we have that $\lim_{t\to+\infty} \sum_{i=0}^{t-1} \alpha^i \tilde{\mathbf{W}}^i = (\mathbf{I} - \alpha\tilde{\mathbf{W}})^{-1}$. Putting these together, we have that

$$\mathbf{s} = \lim_{t\to+\infty} \mathbf{s}^{(t)} = (1-\alpha)(\mathbf{I} - \alpha\tilde{\mathbf{W}})^{-1}\mathbf{l} \tag{3.19}$$

Next, we prove that the above result is indeed the global minimal solution of the objective function defined in Eq. (3.9). We prove this by showing that $J(\mathbf{s})$ in Eq. (3.9) is convex. To see this, we have that the Hessian matrix of Eq. (3.9) is $\nabla^2 J = 2(\mathbf{I} - \alpha\tilde{\mathbf{W}})$ with all eigenvalues of $2(\mathbf{I} - \alpha\tilde{\mathbf{W}})$ greater than 0. In other words, we have that $\nabla^2 J$ is positive definite. Therefore, the objective function defined in Eq. (3.9) is convex, and its fixed-point solution by Algorithm 3.1 corresponds to its global minimal solution, which completes the proof. QED.

The time and space complexity of Algorithm 3.1 are summarized in Lemma 3.2. Notice that such a complexity is comparable to the complexity of topology-alone network alignment methods, such as IsoRank [3]. In the next section, we will introduce an even faster algorithm.

Lemma 3.2. Complexity of FINAL-NE. The time complexity of Algorithm 3.1 is $O(Lmnt_{\max} + LKn^2)$, and its space complexity is $O(n^2)$. Here, n and m are the orders of the number of nodes and edges of the input networks, respectively; K, L denote the dimension of node and edge feature vectors respectively, and $t_{\max}$ is the maximum iteration number.

Proof. It requires $O(nK + mL)$ time and space complexity for line 1-2. To compute $\mathbf{N}$, it takes $O(Kn^2)$ time complexity and $O(n^2)$ space complexity. Then based on Eq. (3.10), constructing $\mathbf{D}$ requires $O(2m + n^2)KL$ time complexity and $O(n^2)$ space complexity. Line 9-11 takes $O(Lmn)$ time complexity and $O(n^2)$ space complexity. Thus, line 5-14 with $t_{\max}$ iterations takes $O(Lmnt_{\max})$ time complexity and $O(n^2)$ space complexity. In total, the FINAL-NE algorithm takes $O(Lmnt_{\max} + LKn^2)$ time complexity and $O(n^2)$ space complexity. This completes the proof. QED.

Finally, we analyze the relationships between the proposed FINAL algorithms and several classic graph mining problems. For the sake of conciseness, we omit the detailed proofs and summarize the major findings as follows.

A - FINAL *vs. Node Proximity.* An important (single) network mining task is the node proximity, i.e., to measure the proximity/similarity between two nodes on the *same* network. By ignoring the node/edge attributes and setting $\mathbf{A}_1 = \mathbf{A}_2$, our FINAL algorithms, up to a scaling operation $\mathbf{D}^{1/2}$, degenerate to SimRank [106] - a prevalent choice for node proximity. Our FINAL algorithms are also closely related to another popular node proximity method, random walk with restart [107]. That is, Eq. (3.13) can be viewed as random walk with restart on the attributed Kronecker graph with $\mathbf{l}$ being the starting vector. Note that neither the standard SimRank nor random walk with restart considers the node or edge attribute information.

B - FINAL *vs. Graph Kernel.* The alignment result $\mathbf{s}$ by our FINAL algorithms is closely related to the random walk based graph kernel [82]. To be specific, if $k(\mathcal{G}_1, \mathcal{G}_2)$ is the random walk graph kernel between the two input graphs and $\mathbf{p}$ is the stopping vector, we can show that $k(\mathcal{G}_1, \mathcal{G}_2) = \mathbf{p}'\mathbf{s}$. This intuitively makes sense, as we can view the graph kernel/similarity as the weighted aggregation (by the stopping vector $\mathbf{p}$) over the pairwise cross-network node similarities (encoded by the alignment vector $\mathbf{s}$). We also remark that in the original random walk graph kernel [82], it mainly focuses on the node attribute information.

C - FINAL *vs. Prior Network Alignment Methods.* If we ignore all the node and edge attribute information, our FINAL-P algorithm is equivalent to IsoRank [3] by scaling the alignment result and alignment preference by $\mathbf{D}^{1/2}$. We would like to point out that such a scaling operation is important to ensure the convergence of the iterative procedure. Recall

that the key idea behind our optimization formulation is the *alignment consistency*. When the attribute information is absent, the *alignment consistency* principle is closely related to the concept of "squares" behind NetAlign algorithm [5]. Like most, if not all of the, prior network alignment algorithms, the node or the edge attribute information is ignored in IsoRank and NetAlign.

We remark that these findings are important in the following two aspects. First, they help establish a quantitative relationship between several, seemingly unrelated graph mining problems, which might in turn help better understand these existing graph mining problems. Second, these findings also have an important algorithmic implication. Take SimRank as an example, it was originally designed for plain graphs (i.e., without attributes), and was formulated from random walk perspective and it is not clear what the algorithm tries to optimize. By setting $\mathcal{G}_1 = \mathcal{G}_2$ and ignoring the attribute information, our objective function in Eq. (3.9) provides a natural way to interpret SimRank from an optimization perspective. By setting $\mathcal{G}_1 = \mathcal{G}_2$ alone, our FINAL algorithms can be directly used to measure node proximity on an attributed network. Finally, our upcoming FINAL On-Query algorithm also naturally provides an efficient way (i.e., with a linear time complexity) for *on-query* SimRank with or without attribute information (i.e., finding the similarity between a given query node and all the remaining nodes in the same network).

3.1.3 Speed-up Computations

In this part, we address the computational issue to handle the volume characteristic of big networks. To be specific, we focus on two scenarios. First, to solve Problem 3.1, our proposed FINAL algorithms have a time complexity of $O(mn)$, where we have dropped the lower order terms. We develop an effective approximate algorithm that reduces the time complexity to $O(n^2)$. Second, for Problem 3.2, solving the full alignment problem not only still requires $O(n^2)$ time, but also is unnecessary, as we essentially only need a column or a row from the alignment matrix $\mathbf{S}$. To address this issue, we present an effective algorithm for Problem 3.2 with a *linear* time complexity. For presentation clarity, we restrict ourselves to the case where there is only node attribute information, although our proposed strategies can be naturally applied to the more general case where we have both node and edge attributes. Moreover, in the scenario that networks have no attributes, we propose a multilevel network alignment method using the *coarsen-align-interpolate* strategy to approximate FINAL-P into a *linear* algorithm Moana [11].

Speed-up FINAL-N. According to Lemma 3.1, the alignment vector $\mathbf{s}$ in FINAL-N converges to its closed-form solution as follows.

$$\begin{aligned}\mathbf{s} &= (1-\alpha)(\mathbf{I} - \alpha\tilde{\mathbf{W}}_N)^{-1}\mathbf{l} \\ &= (1-\alpha)(\mathbf{I} - \alpha\mathbf{D}_N^{-\frac{1}{2}}\mathbf{N}(\mathbf{A}_2 \otimes \mathbf{A}_1)\mathbf{N}\mathbf{D}_N^{-\frac{1}{2}})^{-1}\mathbf{l}\end{aligned} \tag{3.20}$$

The key idea to speed up FINAL-N is to efficiently approximate such a closed-form solution. To be specific, we first approximate the two adjacency matrices by top-r eigenvalue decomposition: $\mathbf{A}_1 = \mathbf{U}_1\mathbf{\Lambda}_1\mathbf{U}_1'$ and $\mathbf{A}_2 = \mathbf{U}_2\mathbf{\Lambda}_2\mathbf{U}_2'$. Then the rank-$r$ approximation of $\mathbf{W}_N$ can be defined as follows

$$\begin{aligned}\hat{\mathbf{W}}_N &= \mathbf{N}[(\mathbf{U}_2\mathbf{\Lambda}_2\mathbf{U}_2') \otimes (\mathbf{U}_1\mathbf{\Lambda}_1\mathbf{U}_1')]\mathbf{N} \\ &= \mathbf{N}(\mathbf{U}_2 \otimes \mathbf{U}_1)(\mathbf{\Lambda}_2 \otimes \mathbf{\Lambda}_1)(\mathbf{U}_2' \otimes \mathbf{U}_1')\mathbf{N}\end{aligned} \tag{3.21}$$

By substituting Eq. (3.21) into Eq. (3.20), we can approximate the alignment vector $\mathbf{s}$ as

$$\begin{aligned}\mathbf{s} \approx& (1-\alpha)[\mathbf{I} - \alpha\mathbf{D}_N^{-\frac{1}{2}}\mathbf{N}\mathbf{U}(\mathbf{\Lambda}_2 \otimes \mathbf{\Lambda}_1)\mathbf{U}'\mathbf{N}\mathbf{D}_N^{-\frac{1}{2}}]^{-1}\mathbf{l} \\ =& (1-\alpha)(\mathbf{I} + \alpha\mathbf{D}_N^{-\frac{1}{2}}\mathbf{N}\mathbf{U}\mathbf{\Lambda}\mathbf{U}'\mathbf{N}\mathbf{D}_N^{-\frac{1}{2}})\mathbf{l}\end{aligned} \tag{3.22}$$

where $\mathbf{U} = \mathbf{U}_2 \otimes \mathbf{U}_1$, and $\mathbf{\Lambda}$ is an $r^2 \times r^2$ matrix computed by Woodbury matrix identity [108]: $\mathbf{\Lambda} = [(\mathbf{\Lambda}_2 \otimes \mathbf{\Lambda}_1)^{-1} - \alpha(\mathbf{U}_2' \otimes \mathbf{U}_1')\mathbf{N}\mathbf{D}_N^{-1}\mathbf{N}(\mathbf{U}_2 \otimes \mathbf{U}_1)]^{-1}$.

Based on Eq. (3.22), our proposed FINAL-N+ algorithm is summarized in Algorithm 3.2. The time complexity of FINAL-N+ is summarized in Lemma 3.3. Notice that we often have $r \ll n$, $m \ll n^2$ and $K \ll n$. Therefore, compared with FINAL-N, FINAL-N+ is much more efficient in time complexity.

Lemma 3.3. Time Complexity of FINAL-N+. FINAL-N+ takes $O(n^2r^4 + Kn^2)$ in time where n is the order of the number of nodes, r is the rank of eigenvalue decomposition and K is the number of node attributes.

Proof. Omitted for space. QED.

Proposed Solution for Problem 3.2. In Problem 3.2, we want to find an $1 \times n_2$ vector $\mathbf{s}_a$ which measures the similarities between the query node-a in $\mathcal{G}_1$ and all the n_2 nodes in $\mathcal{G}_2$ (i.e., cross-network similarity search). It is easy to see that $\mathbf{s}_a$ is essentially the a-th row of the alignment matrix $\mathbf{S}$, or equivalently a certain portion of the alignment vector $\mathbf{s}$, i.e., $\mathbf{s}_a = \mathbf{S}(a,:) = \mathbf{s}(\mathcal{I})$ where $\mathcal{I} = \{(i-1)n_1 + a | i = 1, \cdots, n_2\}$.

Algorithm 3.2: FINAL-N+: Low-Rank Approximation of FINAL-N.

Input : (1) $\mathcal{G}_1 = \{\mathbf{A}_1, \mathbf{X}_1\}$ and $\mathcal{G}_2 = \{\mathbf{A}_2, \mathbf{X}_2\}$, (2) optional prior alignment preference $\mathbf{L}$, (3) the regularization parameter α, and (4) the rank of eigenvalue decomposition r.

Output: approximate alignment matrix $\mathbf{S}$ between $\mathcal{G}_1$ and $\mathcal{G}_2$.

Construct degree matrix $\mathbf{D}_N$ and node attribute matrix $\mathbf{N}$;
Construct alignment preference vector $\mathbf{l} = \text{vec}(\mathbf{L})$;
Eigenvalue decomposition $\mathbf{U}_1\mathbf{\Lambda}_1\mathbf{U}_1' \leftarrow \mathbf{A}_1$, $\mathbf{U}_2\mathbf{\Lambda}_2\mathbf{U}_2' \leftarrow \mathbf{A}_2$;
Compute $\mathbf{U} = \mathbf{U}_2 \otimes \mathbf{U}_1$;
Compute $\mathbf{\Lambda} = [(\mathbf{\Lambda}_2 \otimes \mathbf{\Lambda}_1)^{-1} - \alpha\mathbf{U}'\mathbf{N}\mathbf{D}_N^{-1}\mathbf{N}\mathbf{U}]^{-1}$;
Compute $\mathbf{s}$ by Eq. (3.22);
Reshape vector $\mathbf{s}$ to $\mathbf{S} = \text{mat}(\mathbf{s}, n_1, n_2)$.

However, if we call FINAL-N or FINAL-N+ to find $\mathbf{S}$ (or $\mathbf{s}$) and then return the ranking vector $\mathbf{s}_a$, it would take at least $O(n^2)$ time. Next, we develop an approximate algorithm (FINAL On-Query) which directly finds the ranking vector $\mathbf{s}_a$ in *linear* time, without solving the full alignment matrix $\mathbf{S}$.

We first relax the degree matrix $\mathbf{D}_N$ to its upper-bound $\hat{\mathbf{D}}_N = \mathbf{D}_2 \otimes \mathbf{D}_1$. There are two reasons for taking such a relaxation. First, it would take $O(n^2)$ time to compute the $\mathbf{D}_N$ matrix directly. On the other hand, $\hat{\mathbf{D}}_N$ can be indirectly expressed by the Kronecker product between $\mathbf{D}_1$ and $\mathbf{D}_2$, each of which only takes $O(m)$ time. Second, since $\hat{\mathbf{D}}_N$ is an upper-bound of the $\mathbf{D}_N$ matrix, such a relaxation will not affect the convergence of FINAL-N. By relaxation, the fixed-point solution in Eq. (3.13) can be approximated as

$$\mathbf{s} = \alpha\mathbf{N}\hat{\mathbf{D}}_N^{-\frac{1}{2}}(\mathbf{A}_2 \otimes \mathbf{A}_1)\hat{\mathbf{D}}_N^{-\frac{1}{2}}\mathbf{N}\mathbf{s} + (1-\alpha)\mathbf{l} \tag{3.23}$$

where $\hat{\mathbf{D}}_N = \mathbf{D}_2 \otimes \mathbf{D}_1$. By a similar procedure in FINAL-N+, the low-rank approximate solution for $\mathbf{s}$ is

$$\mathbf{s} \approx (1-\alpha)\mathbf{l} + \alpha(1-\alpha)\hat{\mathbf{D}}_N^{-\frac{1}{2}}\mathbf{N}\mathbf{U}\hat{\mathbf{\Lambda}}\mathbf{U}'\mathbf{N}\hat{\mathbf{D}}_N^{-\frac{1}{2}}\mathbf{l} \tag{3.24}$$

where $\hat{\mathbf{\Lambda}} = [(\mathbf{\Lambda}_2 \otimes \mathbf{\Lambda}_1)^{-1} - \alpha\mathbf{U}'\mathbf{N}\hat{\mathbf{D}}_N^{-1}\mathbf{U}]^{-1}$.

Since both $\hat{\mathbf{D}}_N$ and $\mathbf{N}$ are diagonal matrices, the ranking vector for node-a is

$$\begin{aligned}
\mathbf{s}_a =& (1-\alpha)[\mathbf{l}(\mathcal{I}) + \alpha[\hat{\mathbf{D}}_N^{-\frac{1}{2}}\mathbf{N}\mathbf{U}\hat{\mathbf{\Lambda}}\mathbf{U}'\mathbf{N}\hat{\mathbf{D}}_N^{-\frac{1}{2}}\mathbf{l}](\mathcal{I})] \\
=& (1-\alpha)[\mathbf{L}(a,:) + \alpha[(\mathbf{D}_1(a,a)\mathbf{D}_2)^{-\frac{1}{2}}(\sum_{k=1}^{K}\mathbf{N}_1^k(a,a)\mathbf{N}_2^k)] \\
&\times[\underbrace{(\mathbf{U}_2 \otimes \mathbf{U}_1(a,:))}_{O(nr^2)} \underbrace{\hat{\mathbf{\Lambda}}}_{O(n^2r^4+r^6)} \underbrace{\mathbf{U}'\mathbf{N}\hat{\mathbf{D}}_N^{-\frac{1}{2}}\mathbf{l}}_{O(n^2r^2)}]
\end{aligned} \tag{3.25}$$

Notice that Eq. (3.25) still needs $O(n^2)$ time due to the last two terms. We reduce the time cost for computing $\mathbf{g} = \mathbf{U}'\mathbf{N}\hat{\mathbf{D}}_N^{-\frac{1}{2}}\mathbf{l}$ as follows. First, we take a rank-p singular value decomposition (SVD) on $\mathbf{L}$, i.e., $\mathbf{L} = \sum_{i=1}^{p} \sigma_i \mathbf{u}_i \mathbf{v}_i'$. Then, by the vectorization operator, we have that

$$\mathbf{g} = \sum_{i=1}^{p}\sum_{k=1}^{K} \underbrace{\sigma_i(\overbrace{\mathbf{U}_2'\mathbf{N}_2^k\mathbf{D}_2^{-\frac{1}{2}}\mathbf{v}_i}^{O(nr)}) \otimes (\overbrace{\mathbf{U}_1'\mathbf{N}_1^k\mathbf{D}_1^{-\frac{1}{2}}\mathbf{u}_i}^{O(nr)})}_{O(nr+r^2)=O(nr)} \tag{3.26}$$

We can see that the time cost for Eq. (3.26) is reduced to $O(pKrn)$, which is linear w.r.t the number of nodes n.

We reduce the time cost for computing $\hat{\mathbf{\Lambda}}$ by reformulating as follows, whose time complexity is $O(Knr^2 + Kr^4 + r^6)$

$$\hat{\mathbf{\Lambda}} = [\underbrace{(\mathbf{\Lambda}_2 \otimes \mathbf{\Lambda}_1)^{-1}}_{O(r^2)} - \alpha \sum_{k=1}^{K} \underbrace{(\overbrace{\mathbf{U}_2'\mathbf{N}_2^k\mathbf{D}_2^{-1}\mathbf{U}_2}^{O(nr^2)}) \otimes (\overbrace{\mathbf{U}_1'\mathbf{N}_1^k\mathbf{D}_1^{-1}\mathbf{U}_1}^{O(nr^2)})}_{O(nr^2+r^4)}]^{-1} \tag{3.27}$$

Putting everything together, the ranking vector of node-a now becomes

$$\begin{aligned}\mathbf{s}_a =& (1-\alpha)\mathbf{L}(a,:) + \alpha(1-\alpha)\overbrace{[(\mathbf{D}_1(a,a)\mathbf{D}_2)^{-\frac{1}{2}}}^{O(n)}\overbrace{\sum_{k=1}^{K}\mathbf{N}_1^k(a,a)\mathbf{N}_2^k]}^{O(Kn)} \\ &\times[\underbrace{(\mathbf{U}_2 \otimes \mathbf{U}_1(a,:))}_{O(nr^2)} \underbrace{\hat{\mathbf{\Lambda}}}_{O(Knr^2+Kr^4+r^6)} \underbrace{\mathbf{g}}_{O(pKnr)}]\end{aligned} \tag{3.28}$$

Based on Eq. (3.28), our proposed FINAL On-Query algorithm is summarized in Algorithm 3.3. The time complexity of FINAL On-Query is summarized in Lemma 3.4. Notice that we often have $r, p \ll n$, $m_L \ll m \ll n^2$ and $K \ll n$. FINAL On-Query has a *linear* time complexity w.r.t the size of the input network, which is much more scalable than both FINAL-N and FINAL-N+.

Lemma 3.4. Time complexity of FINAL On-Query. The time complexity of FINAL On-Query is $O(r^6 + mr + nr^2 + m_L p + np^2 + Knr^2 + Kr^4 + pKnr)$ where n, m are the orders of the number of nodes and edges respectively, r, p is the rank of eigenvalue decomposition and SVD, respectively, K is the number of node attributes and m_L is the number of non-zero elements in $\mathbf{L}$.

Proof. Omitted for brevity. QED.

Algorithm 3.3: FINAL On-Query: Approximate On-Query Algorithm for Node Attributed Networks.

Input : (1) $\mathcal{G}_1 = \{\mathbf{A}_1, \mathbf{N}_1\}$ and $\mathcal{G}_2 = \{\mathbf{A}_2, \mathbf{N}_2\}$, (2) optional prior alignment preference $\mathbf{H}$, (3) the regularization parameter α, (4) the rank of eigenvalue decomposition r, and (5) the rank of SVD for $\mathbf{L}$ p.

Output: approximate ranking vector $\mathbf{s}_a$ between node-a in $\mathcal{G}_1$ and all nodes in $\mathcal{G}_2$.

Pre-Compute:

Compute degree matrices $\mathbf{D}_1$ and $\mathbf{D}_2$;

Compute $\mathbf{D}_a = \mathbf{D}_1(a,a)\mathbf{D}_2$, and $\mathbf{N}_a = \sum_{k=1}^{K} \mathbf{N}_1^k(a,a)\mathbf{N}_2^k$;

Rank r eigenvalue decomposition $\mathbf{U}_1\mathbf{\Lambda}_1\mathbf{U}_1' \leftarrow \mathbf{A}_1$;

Rank r eigenvalue decomposition $\mathbf{U}_2\mathbf{\Lambda}_2\mathbf{U}_2' \leftarrow \mathbf{A}_2$;

Rank p singular value decomposition $\sum_{i=1}^{p} \sigma_i \mathbf{u}_i \mathbf{v}_i' \leftarrow \mathbf{L}$;

Compute $\mathbf{g}$ by Eq. (3.26);

Compute $\hat{\mathbf{\Lambda}}$ by Eq. (3.27);

Online-Query:

Compute $\mathbf{s}_a$ by Eq. (3.28).

Speed-up FINAL-P. In this work, we aim to leverage the *hierarchical cluster-within-clusters* characteristics of many real-world networks to not only align at the finest node level, but also align clusters at different coarse levels. The key idea is to coarsen-align-then-interpolate where (1) the first step coarsens the input networks into several smaller networks which depict the supernodes/clusters connections at the coarse level, (2) in the second step we compute the alignment matrix $\mathbf{S}_L$ efficiently at the coarsest level, and then (3) we use the interpolation matrices to efficiently estimate the alignment matrix at the finer level (e.g., $\mathbf{S}_l$ at the l-th level) from that at the next coarser level (e.g., $\mathbf{S}_{l+1}$). Note that since the alignment matrices hinge on two networks, different from the interpolations underlying a single network, the bilinear interpolations are required.

Multilevel Optimization Formulation. To derive the optimization formulation for our multilevel network alignment problem, without loss of generality, we focus on the first two levels for now. Denote two interpolation matrices $\mathbf{P}_1 \in \mathbb{R}^{p_1 \times n_1}$ and $\mathbf{Q}_1 \in \mathbb{R}^{q_1 \times n_2}$ where $p_1 \leq n_1, q_1 \leq n_2$ such that we can approximate the node-level alignment matrix $\mathbf{S}_1$ by $\mathbf{S}_1 = \mathbf{P}_1'\mathbf{S}_2\mathbf{Q}_1$ where $\mathbf{S}_2 \in \mathbb{R}^{p_1 \times q_1}$ is the alignment matrix at the second level. By de-vectorization on $\mathbf{s}_1, \mathbf{l}_1$, Eq. (3.11) without attributes is equivalent to

$$\min_{\mathbf{S}_1} \quad \alpha[\mathrm{Tr}(\mathbf{S}_1'\mathbf{S}_1) - \mathrm{Tr}(\mathbf{S}_1'\tilde{\mathbf{A}}_1^{(1)}\mathbf{S}_1\tilde{\mathbf{A}}_2^{(1)})] + (1-\alpha)\|\mathbf{S}_1 - \mathbf{L}_1\|_F^2 \tag{3.29}$$

where $\tilde{\mathbf{A}}_1^{(1)}, \tilde{\mathbf{A}}_2^{(1)}$ represent the symmetrically normalized adjacency matrices of $\mathcal{G}_1, \mathcal{G}_2$ at the first level and $\mathbf{L}_1 = \mathbf{L}$ denotes the prior alignment matrix at the first level. Plugging in

$\mathbf{S}_1 = \mathbf{P}_1'\mathbf{S}_2\mathbf{Q}_1$, we have the objective function w.r.t. $\mathbf{S}_2$.

$$J(\mathbf{S}_2) = \alpha[\mathrm{Tr}(\mathbf{Q}_1'\mathbf{S}_2'\mathbf{P}_1\mathbf{P}_1'\mathbf{S}_2\mathbf{Q}_1) - \mathrm{Tr}(\mathbf{S}_2'\mathbf{P}_1\tilde{\mathbf{A}}_1^{(1)}\mathbf{P}_1'\mathbf{S}_2\mathbf{Q}_1\tilde{\mathbf{A}}_2^{(1)}\mathbf{Q}_1')] + (1-\alpha)\|\mathbf{P}_1'\mathbf{S}_2\mathbf{Q}_1 - \mathbf{L}_1\|_F^2 \tag{3.30}$$

Notice that if the (semi-) orthogonality satisfies, i.e., $\mathbf{P}_1\mathbf{P}_1' = \mathbf{I}$ and $\mathbf{Q}_1\mathbf{Q}_1' = \mathbf{I}$, we can obtain the objective function at the second level which is of exactly the same form as Eq. (3.29),

$$J(\mathbf{S}_2) = \alpha[\mathrm{Tr}(\mathbf{S}_2'\mathbf{S}_2) - \mathrm{Tr}(\mathbf{S}_2'\tilde{\mathbf{A}}_1^{(2)}\mathbf{S}_2\tilde{\mathbf{A}}_2^{(2)})] + (1-\alpha)\|\mathbf{S}_2 - \mathbf{L}_2\|_F^2 \tag{3.31}$$

where $\tilde{\mathbf{A}}_1^{(2)} = \mathbf{P}_1\tilde{\mathbf{A}}_1^{(1)}\mathbf{P}_1'$, $\tilde{\mathbf{A}}_2^{(2)} = \mathbf{Q}_1\tilde{\mathbf{A}}_2^{(1)}\mathbf{Q}_1'$ and $\mathbf{L}_2 = \mathbf{P}_1\mathbf{L}_1\mathbf{Q}_1'$. Equivalently, this can be viewed as coarsening $\tilde{\mathbf{A}}_1^{(1)}, \tilde{\mathbf{A}}_2^{(1)}$ into $\tilde{\mathbf{A}}_1^{(2)}, \tilde{\mathbf{A}}_2^{(2)}$ to be aligned at the second level by the interpolation matrices $\mathbf{P}_1$ and $\mathbf{Q}_1$, with the corresponding prior node similarity matrix $\mathbf{L}_2$.

Perfect Interpolation. In this work, instead of exploring the semi-orthogonal interpolation matrices, we seek to find a set of orthogonal matrices, i.e., $\mathbf{P}_l\mathbf{P}_l' = \mathbf{P}_l'\mathbf{P}_l = \mathbf{I}$. Indeed, by the following lemma, we show that the orthogonal interpolation matrices guarantee that the interpolation of the optimal alignment matrix from the coarser level is exactly the same as the optimal alignment matrix at the finer level.

Lemma 3.5. Perfect Interpolation. The global optimal solution to the optimization problem at the finer level (e.g., Eq. (3.29) for level-1), denoted by $\mathbf{S}_l$, is exactly same as the interpolation of the optimal solution at the next coarser level (denoted by $\mathbf{S}_{l+1}$). That is, $\mathbf{S}_l = \mathbf{P}_l'\mathbf{S}_{l+1}\mathbf{Q}_l$ if $\mathbf{P}_l$ and $\mathbf{Q}_l$ are orthogonal, where $l = 1, \cdots, L-1$.

Proof. Without loss of generality, we prove $\mathbf{S}_1 = \mathbf{P}_1'\mathbf{S}_2\mathbf{Q}_1$. The optimal closed-form solution to Eq. (3.11) without attributes (and equivalently Eq. (3.29)) is $\mathbf{s}_1 = (1-\alpha)(\mathbf{I} - \alpha\tilde{\mathbf{A}}_2^{(1)} \otimes \tilde{\mathbf{A}}_1^{(1)})^{-1}\mathbf{l}_1$. Similarly, the alignment matrix between $\tilde{\mathbf{A}}_1^{(2)}, \tilde{\mathbf{A}}_2^{(2)}$ at the second level is computed by

$$\mathbf{s}_2 = (1-\alpha)[\mathbf{I} - \alpha(\mathbf{Q}_1\tilde{\mathbf{A}}_2^{(1)}\mathbf{Q}_1') \otimes (\mathbf{P}_1\tilde{\mathbf{A}}_1^{(1)}\mathbf{P}_1')]^{-1}\mathbf{l}_2 \tag{3.32}$$

The difference between $\mathbf{S}_1$ and the interpolated alignment from $\mathbf{S}_2$ in the Frobenius norm is

$$\begin{aligned}
&\|\mathbf{P}_1'\mathbf{S}_2\mathbf{Q}_1 - \mathbf{S}_1\|_F = \|(\mathbf{Q}_1' \otimes \mathbf{P}_1')\mathbf{s}_2 - \mathbf{s}_1\|_2 \\
&= (1-\alpha)\big\|(\mathbf{Q}_1' \otimes \mathbf{P}_1')[\mathbf{I} - \alpha(\mathbf{Q}_1\tilde{\mathbf{A}}_2^{(1)}\mathbf{Q}_1') \otimes (\mathbf{P}_1\tilde{\mathbf{A}}_1^{(1)}\mathbf{P}_1')]^{-1}\mathbf{l}_2 - (\mathbf{I} - \alpha\tilde{\mathbf{A}}_2^{(1)} \otimes \tilde{\mathbf{A}}_1^{(1)})^{-1}\mathbf{l}_1\big\|_2 \\
&= (1-\alpha)\big\|\sum_{k=0}^{\infty}\alpha^k(\mathbf{Q}_1' \otimes \mathbf{P}_1')[(\mathbf{Q}_1\tilde{\mathbf{A}}_2^{(1)}\mathbf{Q}_1')^k \otimes (\mathbf{P}_1\tilde{\mathbf{A}}_1^{(1)}\mathbf{P}_1')^k](\mathbf{Q}_1 \otimes \mathbf{P}_1)\mathbf{l}_1 \\
&- \sum_{k=0}^{\infty}\alpha^k[(\tilde{\mathbf{A}}_2^{(1)})^k \otimes (\tilde{\mathbf{A}}_1^{(1)})^k]\mathbf{l}_1\big\|_2
\end{aligned} \tag{3.33}$$

where the second equation is by Neumann series due to the fact that (1) the eigenvalues of $\tilde{\mathbf{A}}_1^{(1)}, \tilde{\mathbf{A}}_2^{(1)}$ are in the range of $(-1, 1)$, and (2) $\mathbf{P}_1\tilde{\mathbf{A}}_1^{(1)}\mathbf{P}_1', \mathbf{Q}_1\tilde{\mathbf{A}}_2^{(1)}\mathbf{Q}_1'$ share the same eigenvalues as $\tilde{\mathbf{A}}_1^{(1)}, \tilde{\mathbf{A}}_2^{(1)}$ respectively given that $\mathbf{P}_1$ and $\mathbf{Q}_1$ are orthogonal [108].

Due to the orthogonality of $\mathbf{P}_1, \mathbf{Q}_1$, the following equations hold.

$$(\tilde{\mathbf{A}}_1^{(1)})^k = \mathbf{P}_1'(\mathbf{P}_1\tilde{\mathbf{A}}_1^{(1)}\mathbf{P}_1')^k\mathbf{P}_1, \quad (\tilde{\mathbf{A}}_2^{(1)})^k = \mathbf{Q}_1'(\mathbf{Q}_1\tilde{\mathbf{A}}_2^{(1)}\mathbf{Q}_1')^k\mathbf{Q}_1 \tag{3.34}$$

Thus, we have that

$$\|\mathbf{S}_1 - \mathbf{P}_1'\mathbf{S}_2\mathbf{Q}_1\|_F^2 = (1-\alpha)\big\|\sum_{k=0}^{\infty}\alpha^k[(\tilde{\mathbf{A}}_2^{(1)})^k \otimes (\tilde{\mathbf{A}}_1^{(1)})^k - (\tilde{\mathbf{A}}_2^{(1)})^k \otimes (\tilde{\mathbf{A}}_1^{(1)})^k]\mathbf{l}_1\big\|_2 = 0 \tag{3.35}$$

which completes the proof. QED.

Multilevel Alignment Algorithm. We seek to find a set of orthogonal interpolation matrices $\mathbf{P}_l$ and $\mathbf{Q}_l$ such that (1) they are sufficiently sparse, and (2) they are able to uncover the *hierarchical cluster-within-clusters* structure of the input networks. In this work, we leverage the multiresolution matrix factorization (MMF) algorithm that satisfies these requirements [109]. Specifically, for each input network, we use the parallel second order MMF algorithm to find a set of rotation matrices $\mathbf{P}_l, \mathbf{Q}_l$ such that at the l-th level ($l \geq 2$),

$$\tilde{\mathbf{A}}_1^{(l)} = \mathbf{P}_{l-1}\cdots\mathbf{P}_1\tilde{\mathbf{A}}_1^{(1)}\mathbf{P}_1'\cdots\mathbf{P}_{l-1}' \tag{3.36}$$

$$\tilde{\mathbf{A}}^{(l)} = \mathbf{Q}_{l-1}\cdots\mathbf{Q}_1\tilde{\mathbf{A}}_2^{(1)}\mathbf{Q}_1'\cdots\mathbf{Q}_{l-1}' \tag{3.37}$$

where the active indices of $\tilde{\mathbf{A}}_1^{(l)}, \tilde{\mathbf{A}}_2^{(l)}$ are denoted as $\mathcal{S}_{A_1^{(l)}}$ of size λ_l and $\mathcal{S}_{A_2^{(l)}}$ of size μ_l, respectively. Specifically, at the coarsest level, the rotated matrices are denoted as $\tilde{\mathbf{A}}_1^{(L)}, \tilde{\mathbf{A}}_2^{(L)}$. Then we form the core-diagonal matrices $\bar{\mathbf{A}}_1^{(L)}$ and $\bar{\mathbf{A}}_2^{(L)}$ as in [109].

After the coarsening step, the symmetrically normalized adjacency matrices of the input networks are transformed into the corresponding core-diagonal matrices, i.e.,

$$\bar{\mathbf{A}}_1^{(L)} = \mathbf{\Pi}_{A_1}\begin{bmatrix}\bar{\mathbf{A}}_1^{(L_1)} & \mathbf{0}\\ \mathbf{0} & \bar{\mathbf{A}}_1^{(L_2)}\end{bmatrix}\mathbf{\Pi}_{A_1}', \quad \bar{\mathbf{A}}_2^{(L)} = \mathbf{\Pi}_{A_2}\begin{bmatrix}\bar{\mathbf{A}}_2^{(L_1)} & \mathbf{0}\\ \mathbf{0} & \bar{\mathbf{A}}_2^{(L_2)}\end{bmatrix}\mathbf{\Pi}_{A_2}' \tag{3.38}$$

where $\bar{\mathbf{A}}_1^{(L_1)} = \bar{\mathbf{A}}_1^{(L)}(\mathcal{S}_{A_1^{(L)}}, \mathcal{S}_{A_1^{(L)}})$ and $\bar{\mathbf{A}}_2^{(L_1)} = \bar{\mathbf{A}}_2^{(L)}(\mathcal{S}_{A_2^{(L)}}, \mathcal{S}_{A_2^{(L)}})$ are the core matrices of $\bar{\mathbf{A}}_1^{(L)}$ and $\bar{\mathbf{A}}_2^{(L)}$ respectively. $\mathbf{\Pi}_{A_1}, \mathbf{\Pi}_{A_2}$ are the orthogonal permutation matrices to reorder the active indices of the matrices $\bar{\mathbf{A}}_1^{(L)}, \bar{\mathbf{A}}_2^{(L)}$ to be in the upper left part for the illustration purpose. Denote the inactive indices as $\overline{\mathcal{S}}_{A_1^{(L)}} = \{1, \cdots, n_1\} \setminus \mathcal{S}_{A_1^{(L)}}$ and $\overline{\mathcal{S}}_{A_2^{(L)}} = \{1, \cdots, n_2\} \setminus$

$\mathcal{S}_{A_2^{(L)}}$. Accordingly, $\bar{\mathbf{A}}_1^{(L_2)} = \bar{\mathbf{A}}_1^{(L)}(\overline{\mathcal{S}}_{A_1^{(L)}}, \overline{\mathcal{S}}_{A_1^{(L)}})$ and $\bar{\mathbf{A}}_2^{(L_2)} = \bar{\mathbf{A}}_2^{(L)}(\overline{\mathcal{S}}_{A_2^{(L)}}, \overline{\mathcal{S}}_{A_2^{(L)}})$. Note that our algorithm does not need to explicitly compute such permutation matrices.

We compute the alignment between $\bar{\mathbf{A}}_1^{(L)}$ and $\bar{\mathbf{A}}_2^{(L)}$ at the coarsest level iteratively as

$$\mathbf{S}_L = \alpha \bar{\mathbf{A}}_1^{(L)} \mathbf{S}_L \bar{\mathbf{A}}_2^{(L)} + (1-\alpha)\mathbf{L}_L \tag{3.39}$$

where $\mathbf{L}_L = \mathbf{P}_{L-1} \cdots \mathbf{P}_1 \mathbf{L}_1 \mathbf{Q}_1' \cdots \mathbf{Q}_{L-1}'$ is the corresponding prior similarity matrix at the coarsest level. By using the permutation matrices $\mathbf{\Pi}_{A_1}, \mathbf{\Pi}_{A_2}$, Eq. (3.39) can be rewritten as

$$\mathbf{\Pi}_{A_1}' \mathbf{S}_L \mathbf{\Pi}_{A_2} = \alpha(\mathbf{\Pi}_{A_1}' \bar{\mathbf{A}}_1^{(L)} \mathbf{\Pi}_{A_1})(\mathbf{\Pi}_{A_1}' \mathbf{S}_L \mathbf{\Pi}_{A_2})(\mathbf{\Pi}_{A_2}' \bar{\mathbf{A}}_2^{(L)} \mathbf{\Pi}_{A_2}') + (1-\alpha)\mathbf{\Pi}_{A_1}' \mathbf{L}_L \mathbf{\Pi}_{A_2} \tag{3.40}$$

By denoting $\bar{\mathbf{S}}_L = \mathbf{\Pi}_{A_1}' \mathbf{S}_L \mathbf{\Pi}_{A_2}$ and $\bar{\mathbf{L}}_L = \mathbf{\Pi}_{A_1}' \mathbf{L}_L \mathbf{\Pi}_{A_2}$, the computation can be simplified to

$$\bar{\mathbf{S}}_L = \alpha \begin{bmatrix} \bar{\mathbf{A}}_1^{(L_1)} & \mathbf{0} \\ \mathbf{0} & \bar{\mathbf{A}}_2^{(L_2)} \end{bmatrix} \begin{bmatrix} \bar{\mathbf{S}}_{L_1} & \bar{\mathbf{S}}_{L_2} \\ \bar{\mathbf{S}}_{L_3} & \bar{\mathbf{S}}_{L_4} \end{bmatrix} \begin{bmatrix} \bar{\mathbf{A}}_2^{(L_1)} & \mathbf{0} \\ \mathbf{0} & \bar{\mathbf{A}}_2^{(L_2)} \end{bmatrix} + (1-\alpha) \begin{bmatrix} \bar{\mathbf{L}}_{L_1} & \bar{\mathbf{L}}_{L_2} \\ \bar{\mathbf{L}}_{L_3} & \bar{\mathbf{L}}_{L_4} \end{bmatrix} \tag{3.41}$$

which allows the computation to be block-wise as follows.

$$\bar{\mathbf{S}}_{L_1} = \alpha \bar{\mathbf{A}}_1^{(L_1)} \bar{\mathbf{S}}_{L_1} \bar{\mathbf{A}}_2^{(L_1)} + (1-\alpha)\bar{\mathbf{L}}_{L_1} \tag{3.42}$$

$$\bar{\mathbf{S}}_{L_2} = \alpha \bar{\mathbf{A}}_1^{(L_1)} \bar{\mathbf{S}}_{L_2} \bar{\mathbf{A}}_2^{(L_2)} + (1-\alpha)\bar{\mathbf{L}}_{L_2} \tag{3.43}$$

$$\bar{\mathbf{S}}_{L_3} = \alpha \bar{\mathbf{A}}_1^{(L_2)} \bar{\mathbf{S}}_{L_3} \bar{\mathbf{A}}_2^{(L_1)} + (1-\alpha)\bar{\mathbf{L}}_{L_3} \tag{3.44}$$

$$\bar{\mathbf{S}}_{L_4} = \alpha \bar{\mathbf{A}}_1^{(L_2)} \bar{\mathbf{S}}_{L_4} \bar{\mathbf{A}}_2^{(L_2)} + (1-\alpha)\bar{\mathbf{L}}_{L_4} \tag{3.45}$$

Armed with the iterative fixed-point algorithm, the global optimal solutions to Eq. (3.42), Eq. (3.43) and Eq. (3.44) can be achieved and are denoted as $\bar{\mathbf{S}}_{L_1}^*, \bar{\mathbf{S}}_{L_2}^*, \bar{\mathbf{S}}_{L_3}^*$ respectively. Furthermore, since both $\bar{\mathbf{A}}_1^{(L_2)}$ and $\bar{\mathbf{A}}_2^{(L_2}$ are sparse diagonal matrices, the closed-form optimal solution of Eq. (3.45) can be easily computed by

$$\bar{\mathbf{s}}_{L_4}^* = (1-\alpha)(\mathbf{I} - \alpha \bar{\mathbf{A}}_2^{(L_2)} \otimes \bar{\mathbf{A}}_1^{(L_2)})^{-1} \bar{\mathbf{l}}_{L_4} \tag{3.46}$$

where $\bar{\mathbf{s}}_{L_4}^* = \text{vec}(\bar{\mathbf{S}}_{L_4}^*), \bar{\mathbf{l}}_{L_4} = \text{vec}(\bar{\mathbf{L}}_{L_4})$ and the operator $\otimes$ represents the Kronecker product. In this way, the optimal solution to the alignment problem at the coarsest level $\mathbf{S}_L^*$ is composed of $\mathbf{S}_L^*(\mathcal{S}_{A_1^{(L)}}, \mathcal{S}_{A_2^{(L)}})$, $\mathbf{S}_L^*(\mathcal{S}_{A_1^{(L)}}, \overline{\mathcal{S}}_{A_2^{(L)}})$, $\mathbf{S}_L^*(\overline{\mathcal{S}}_{A_1^{(L)}}, \mathcal{S}_{A_2^{(L)}})$ and $\mathbf{S}_L^*(\overline{\mathcal{S}}_{A_1^{(L)}}, \overline{\mathcal{S}}_{A_2^{(L)}})$ where

$$\begin{aligned} &\mathbf{S}_L^*(\mathcal{S}_{A_1^{(L)}}, \mathcal{S}_{A_2^{(L)}}) = \bar{\mathbf{S}}_{L_1}^*, && \mathbf{S}_L^*(\mathcal{S}_{A_1^{(L)}}, \overline{\mathcal{S}}_{A_2^{(L)}}) = \bar{\mathbf{S}}_{L_2}^* \\ &\mathbf{S}_L^*(\overline{\mathcal{S}}_{A_1^{(L)}}, \mathcal{S}_{A_2^{(L)}}) = \bar{\mathbf{S}}_{L_3}^*, && \mathbf{S}_L^*(\overline{\mathcal{S}}_{A_1^{(L)}}, \overline{\mathcal{S}}_{A_2^{(L)}}) = \bar{\mathbf{S}}_{L_4}^* \end{aligned} \tag{3.47}$$

Algorithm 3.4: Multilevel Network Alignment (MOANA).

Input : (1) the adjacency matrices $\mathbf{A}, \mathbf{A}_2$ of $\mathcal{G}_1,\ \mathcal{G}_2$, (2) the sparse prior alignment preference $\mathbf{L}$, (3) the number of levels L, (4) the parameters α, K.

Output: the alignment matrices $\mathbf{S}_l^*,\ l = 1, \cdots, L$ between $\mathcal{G}_1, \mathcal{G}_2$.

Compute $\tilde{\mathbf{A}}_1, \tilde{\mathbf{A}}_2$ by symmetrically normalizing $\mathbf{A}_1, \mathbf{A}_2$;

Network coarsening:

$\mathbf{P}_1, \cdots, \mathbf{P}_{L-1}$ and $\bar{\mathbf{A}}_1^{(L)} \leftarrow \text{MMF}(\tilde{\mathbf{A}}_1)$;

$\mathbf{Q}_1, \cdots, \mathbf{Q}_{L-1}$ and $\bar{\mathbf{A}}_2^{(L)} \leftarrow \text{MMF}(\tilde{\mathbf{A}}_2)$;

Compute the coarsest level $\mathbf{L}_L = \mathbf{P}_{L-1} \cdots \mathbf{P}_1 \mathbf{L} \mathbf{Q}_1' \cdots \mathbf{Q}_{L-1}'$;

Alignment at the coarsest level:

while *not converged* **do**

  Update $\bar{\mathbf{S}}_{L_1}, \bar{\mathbf{S}}_{L_2}, \bar{\mathbf{S}}_{L_3}$ by Eq. (3.42)-(3.44);

end

Compute $\bar{\mathbf{s}}_{L_4}^*$ by Eq. (3.46) and $\bar{\mathbf{S}}_{L_4}^* = \text{mat}(\bar{\mathbf{s}}_{L_4}^*)$;

Compose $\mathbf{S}_L^*$ by Eq. (3.47);

Alignment interpolation:

Preserve top-K elements in each row/column of $\mathbf{S}_L^*$;

for $l = L - 1 \to 1$ **do**

  Compute $\mathbf{S}_l^* = \mathbf{P}_l' \mathbf{S}_{l+1}^* \mathbf{Q}_l$;

end

After the optimal alignment matrix $\mathbf{S}_L^*$ at the coarsest level is achieved, the alignment at each level $\mathbf{S}_l^*$ can be computed by the interpolation, i.e., $\mathbf{S}_l^* = \mathbf{P}_l' \mathbf{S}_{l+1}^* \mathbf{Q}_l,\ l = 1, \cdots, L-1$. The overall algorithm is summarized in Algorithm 3.4. The complexity analysis of the algorithm is presented in Lemma 3.6, which implies the *linear* time complexity w.r.t. the number of edges in the networks.

Lemma 3.6. Complexity analysis. The time complexity of Algorithm 3.4 is $O(mL + nd_L^2 t_{\max} + L^2 m_L + LKn)$ and its space complexity is $O(L^2 m_L + L^2 Kn + nd_L)$. Here, m, n are the number of edges and nodes in the networks, $d_L = \max(\lambda_L, \mu_L)$ is the size of core matrix. $t_{\max}$ is the number of iterations until convergence in the alignment phase and K is used for top-K preservation of $\mathbf{S}_L^*$. m_L is the number of nonzero elements in the matrix $\mathbf{L}$ and L is the number of levels.

Proof. Omitted for brevity. Proofs can be referred to [11]. QED.

3.1.4 Experimental Evaluations

In this part, we present the experimental results and analysis of our proposed algorithms FINAL and MOANA. The experiments are designed to evaluate the following aspects:

- *Effectiveness*: How accurate are our algorithms for aligning attributed networks?
- *Efficiency*: How fast are our proposed algorithms?

Experimental Setup. We first introduce the experimental setups as follows.
Datasets.

We evaluate our proposed algorithms on eight real-world attributed networks.

- *Co-Authorship Network:* This dataset contains 42,252 nodes and 210,320 edges [110]. Each author has a feature vector which represents the number of publications of the author in each of 29 major conferences.
- *Douban:* This Douban dataset was collected in 2010 and contains $50k$ users and $5M$ edges [111]. Each user has rich information, such as the location and offline event participation. Each edge has an attribute representing whether two users are contacts or friends.
- *Flickr:* This dataset was collected in 2014 and consists of 215,495 users and 9,114,557 friend relationships. Users have detailed profile information, such as gender, hometown and occupation, each of which can be treated as the node attributes [7].
- *Lastfm:* This dataset was collected in 2013 and contains 136,420 users and 1,685,524 following relationships [7]. A detailed profile of some users is also provided, including gender, age and location, etc.
- *Myspace:* This dataset contains 854,498 users and 6,489,736 relationships. The profile of users includes gender, hometown and religion, etc. [7].
- *ACM Citation:* This dataset was collected in 2016 and it contains 2,381,688 papers. Each paper has a list of authors as well as the venue of the paper [7].
- *DBLP Citation:* This dataset was collected in 2016 and it contains 3,272,991 papers. Each paper has a list of authors as well as its venue [7].
- *ArnetMiner:* ArnetMiner dataset consists of the information up to year 2013. The whole dataset has 1,053,188 nodes and 3,916,907 undirected edges [7].
- *Gr-Qc network:* This collaboration network contains 5,241 nodes and 11,923 edges. Each node represents an author, and there exists an edge if two authors have coauthored together [112].

Based on these datasets, we construct the following six alignment scenarios for evaluations.
S1. Co-Authorship vs. Co-Authorship. We extract a subgraph with 9,143 users/nodes from the original dataset, together with their publications in each conference. We randomly permute this subgraph with noisy edge weights and treat it as the second network. We choose the most active conference of a given author as the node attribute, i.e., the conference with the most publications. We construct the prior alignment preference $\mathbf{L}$ based on the node degree similarity. For this scenario, the prior alignment matrix $\mathbf{L}$ alone leads to a very poor alignment result, with only 0.6% one-to-one alignment accuracy.
S2. Douban Online vs. Douban Offline. We construct an alignment scenario for *Douban* dataset in the same way as [111]. We construct the offline network according to users' co-occurrence in social gatherings. We treat people as (1) 'contacts' of each other if they participate in the same offline events more than ten times but less than twenty times, and (2) 'friends' if they co-participate in more than twenty social gatherings. The constructed offline network has 1,118 users and we extract a subgraph with 3,906 nodes from the provided online network that contains all these offline users. We treat the location of a user as the node attribute, and 'contacts'/'friends' as the edge attribute. We use the degree similarity to construct the prior alignment preference $\mathbf{L}$ which itself leads to 7.07% one-to-one alignment accuracy.
S3. Flickr vs. Lastfm. We have the partial ground-truth alignment for these two datasets [7]. We extract the subgraphs from them that contain the given ground-truth nodes. The two subgraphs have 12,974 nodes and 15,436 nodes, respectively. We consider the gender of a user as node attribute. For those users with the missing information of gender, we treat them as 'unknown'. Same as [7], we sort nodes by their pagerank scores and label 1% highest nodes as 'opinion leaders', the next 10% nodes as 'middle class' and remaining nodes as 'ordinary users'. Edges are attributed by the level of people they connect to (e.g., leader with leader). We use the username similarity as the prior alignment preference by the Jaro-Winkler distance [113]. The username similarity alone can correctly align 61.50% users.
S4. Flickr-Myspace. We have the partial ground-truth alignment for these two datasets. We extract two subnetworks that contain these ground-truth nodes. The subgraph of *Flickr* has 6,714 nodes and the subgraph of *Myspace* has 10,733 nodes. We use the same way as *S3* for node attributes, edge attributes and the prior alignment preference. The username similarity achieves 61.80% accuracy.
S5. ACM-DBLP Co-authorship. We extract from both datasets the papers that are published in four areas, including data mining, machine learning, database and information retrieval/web mining. We construct the co-authorship networks based on each paper's co-

author relationship. That is, if two authors co-author a paper in above areas, then we link this two authors in the co-authorship network. Then, we extract from the two constructed co-authorship networks the subgraphs that contain 9,872 nodes and 39,561 edges in ACM co-authorship network and 9,916 nodes and 44,808 edges in DBLP co-authorship network, respectively. Besides, we consider both numerical and categorical attributes. For numerical node attributes, we treat the number of papers of an author in each of the 17 venues as an attribute, which leads to a 17 dimensional feature vector. We use the four areas as the node categorical attributes. That is, for example, if an author published the most papers in data mining area, then we label this node as 'data mining'. We consider categorical edge attributes, and similarly, we use the area where two authors mostly collaborate with each other. We use the degree similarity matrix as the prior alignment preference which alone can only correctly align 20.76% users.

S5. ArnetMiner-ArnetMiner. We use the same method as *S1* to construct the alignment scenario as well as the prior alignment preference. This scenario contains the largest networks, and therefore is used for scalability evaluations.

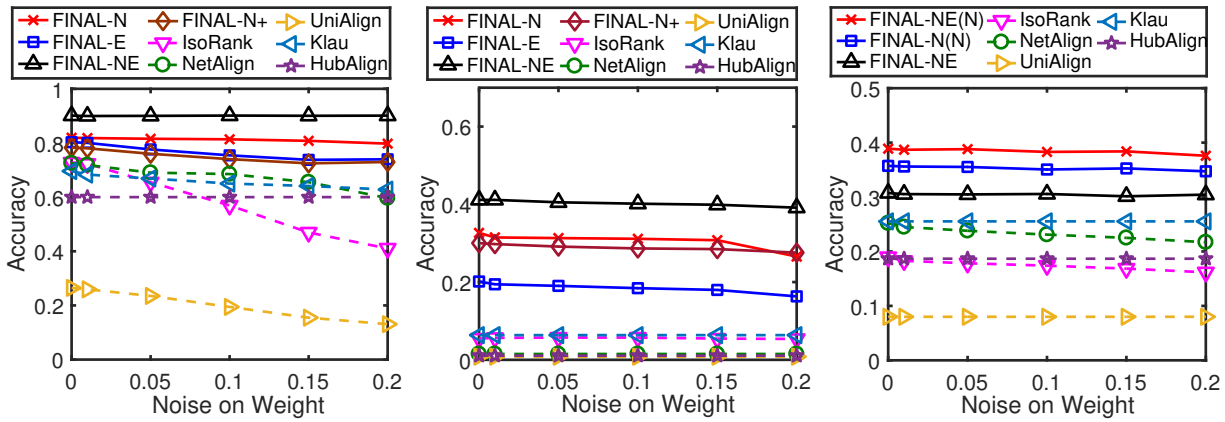
S6. Gr-Qc vs. Gr-Qc. We use the same method as *S1* to construct the alignment scenario as well as the prior alignment preference. We evaluate the efficacy of MOANA on this dataset.

Comparison Methods. For the proposed FINAL algorithms, we test the following variants, including (1) FINAL-NE with categorical node and edge attributes; (2) FINAL-NE(N) with numerical node and edge attributes; (3) FINAL-N with categorical node attributes; (4) FINAL-N(N) with numerical node attributes; (5) FINAL-E with categorical edge attributes; (6) FINAL-N+, a low-rank approximation of FINAL-N. We compare them with the following prior network alignment algorithms including (1) IsoRank [3], (2) NetAlign [5], (3) UniAlign [18], (4) Klau's Algorithm [105], (5) HubAlign [114] and (6) RRWM [115].

Machines and Repeatability. Experiments are performed on a Windows machine with four 3.6GHz Intel Cores and 32G RAM. The algorithms are programmed with MATLAB[1].

Effectiveness Analysis of FINAL. We first evaluate the impact of the permutation noise on the alignment accuracy. We use a heuristic greedy matching algorithm [116] as a post-processing step on the similarity matrix to obtain the one-to-one alignments between the two input networks, and then compute the alignment accuracy with respect to the ground-truth. The results are summarized in Figure 3.3. We have the following observations. First, all of our proposed methods outperform the prior alignment methods. Specifically, our FINAL algorithms achieve an up to 30% improvement in terms of the alignment accu-

[1]The source code of our algorithms can be downloaded here: `https://github.com/sizhang92/FINAL-KDD16`.



(a) DBLP co-author. (b) Douban. (c) ACM-DBLP.

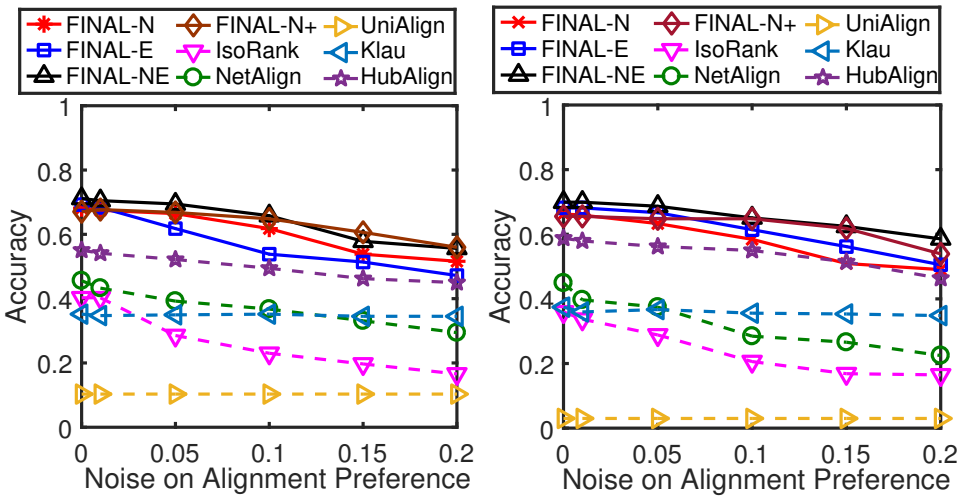
Figure 3.3: (Higher is better.) Alignment accuracy vs. the noise level in networks.

Table 3.2: Alignment with different alignment prior matrices.

	FINAL-NE	FINAL-N	RRWM	IsoRank			NetAlign			Klau			HubAlign		
				-O	-N	-H	-O	-N	-H	-O	-N	-H	-O	-N	-H
DBLP-Coauthor	**0.901**	**0.819**	0.873	0.728	0.810	0.557	0.725	0.712	0.818	0.696	0.480	0.507	0.620	0.631	0.612
Douban	**0.411**	**0.327**	0.042	0.055	0.140	0.305	0.015	0.207	0.024	0.063	0.230	0.056	0.010	0.021	0.053
ACM-DBLP	**0.389**	**0.357**	0.330	0.190	0.178	0.311	0.252	0.288	0.337	0.255	0.199	0.258	0.187	0.133	0.261
Flickr-Lastfm	**0.711**	**0.681**	0.642	0.403	0.299	0.635	0.456	0.235	0.538	0.352	0.283	0.516	0.588	0.310	0.622
Flickr-Myspace	**0.699**	**0.663**	0.678	0.360	0.251	0.543	0.449	0.206	0.371	0.375	0.202	0.408	0.5506	0.157	0.431

racy over the comparison methods. Second, FINAL-N and FINAL-E both outperform the prior methods in most scenarios, yet are not as good as FINAL-NE, suggesting that node attributes and edge attributes might be complementary in terms of improving the alignment accuracy. Third, the alignment accuracy of FINAL-N+ is very close to its exact counterpart FINAL-N (i.e., with a 95% accuracy compared with FINAL-N). Fourth, by jointly considering the attributes and the topology of networks, our methods are more resilient to the permutation noise. Moreover, for the two networks whose topologies are dramatically different from each other (e.g., Douban online-offline networks), the accuracy gap between FINAL-N+ and the prior methods is even bigger (Figure 3.3 (b)). This is because in this case, the topology information alone (IsoRank, NetAlign, Klau and HubAlign) could actually mislead the alignment process. Finally, as Figure 3.3 (c) shows, using numerical attributes (i.e., FINAL-NE(N) and FINAL-N(N)) could further improve the performance of FINAL-NE with categorical attributes.

Second, we evaluate the impact of the noise in the prior alignment preference (i.e., $\mathbf{L}$) on the alignment results, which is summarized in Figure 3.4. As expected, a higher noise in $\mathbf{L}$ has more negative impacts on the alignment accuracy for most of the methods. Nonetheless, our FINAL algorithms still consistently outperform all other four prior methods across different noise levels.

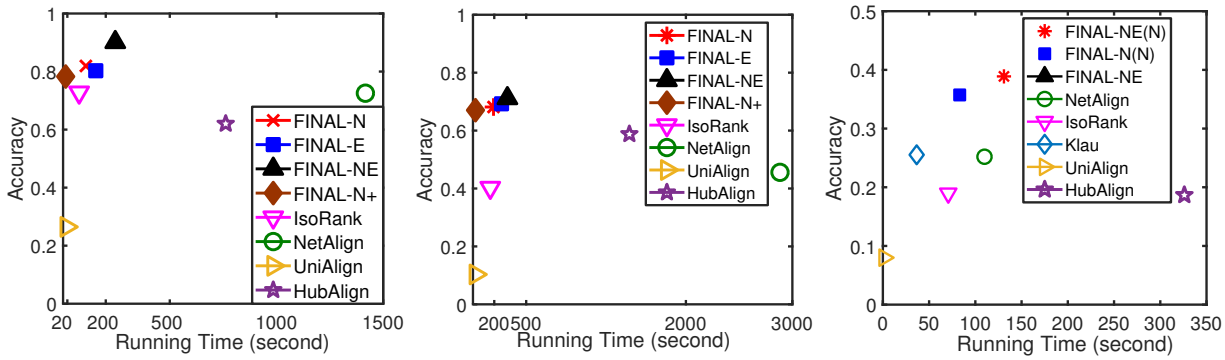


(a) Flickr-Lastfm. (b) Flickr-Myspace.

Figure 3.4: (Higher is better.) Alignment accuracy vs. the noise level in $\mathbf{L}$.

In addition, we conduct the comparisons between the proposed FINAL algorithm and other baseline methods to show the alignment performance by using various information. For RRWM, we compute the cosine similarity matrix of the node attributes and edge attributes respectively, then combine them with the Kronecker product of the adjacency matrices to form the affinity matrix [33]. For the rest of comparison scenarios, we calculate the cosine similarity values among node attributes as the node similarities across networks, which will then be used as the prior alignment matrix $\mathbf{L}$ of the baseline methods (named as -N). Similarly, the average between the node attribute similarity matrix and the originally designed prior matrix (used in Figure 3.3 and Figure 3.4), is considered as the prior matrix (named as -H). Note that the baseline methods using the provided prior alignment matrix $\mathbf{L}$ as aforementioned are named as -O. The results are summarized in Table 3.2. First, we observe that given the exact same set of information, the proposed FINAL-NE outperforms the RRWM algorithm in terms of the alignment accuracy. This indicates even with the more rigorous constraints of the optimization problem in RRWM, the intricacy of solving the problem itself might mislead the alignment solution. Second, we observe that given the node attributes and prior alignment matrix, our proposed FINAL-N method outperforms other baseline methods. This indicates although the additive combination of attributes and the prior knowledge could lead to an improvement within the baseline methods themselves, our algorithms still achieve a better performance due to the *alignment consistency*.

Efficiency Analysis of FINAL. We first evaluate how different methods balance the alignment accuracy and the running time for the full network alignment problem (i.e., Prob-



(a) DBLP co-author. (b) Flickr-Lastfm. (c) ACM-DBLP co-author.

Figure 3.5: Balance between the accuracy and the speed.

lem 3.1). The results are summarized in Figure 3.5. Note that the results of RRWM are not included here because it takes days to finish the computation, which is not even comparable with other methods. As we can see, the running time of our proposed exact methods is only slightly higher than its topology-alone counterpart (i.e., IsoRank), and in the meanwhile, they all achieve a 10%-20% accuracy improvement. FINAL-NE(N) and FINAL-N(N) are faster than FINAL-NE in Figure 3.5 (c) because using numerical attributes could lead to a faster convergence. Besides, FINAL-N+ and UniAlign are the fastest, yet the proposed FINAL-N+ produces a much higher alignment accuracy. We do not show the balance of Klau's Algorithm in Figure 3.5 (a) and (b), because the running time is usually several hours which is not comparable with other methods. For NetAlign and Klau's Algorithm, we observe that they take much longer running time when the input prior alignment preference matrix $\mathbf{L}$ is not sparse enough, as it involves a time-consuming Hungarian step during each iteration.

Second, we evaluate the quality-speed trade-off for on-query alignment problem. Here, we treat the top-10 ranking results by FINAL-N as the ground-truth, and compare the average ranking accuracy of 500 random nodes with two proposed approximate algorithms (FINAL-N+ and FINAL On-Query). The results are summarized in Figure 3.6. We observe that (1) FINAL-N+ preserves a 95% ranking accuracy, with a more than 10× speedup over FINAL-N, (2) FINAL On-Query preserves an about 90% ranking accuracy, and it is 100× faster than the exact FINAL-N.

Scalability of FINAL. We first evaluate the scalability of FINAL-N+, which is summarized in Figure 3.7. We can see that the running time is quadratic w.r.t the number of nodes of the input networks, which is consistent with the time complexity results in Lemma 3.3. Second, we evaluate the scalability of FINAL On-Query, for both its pre-compute

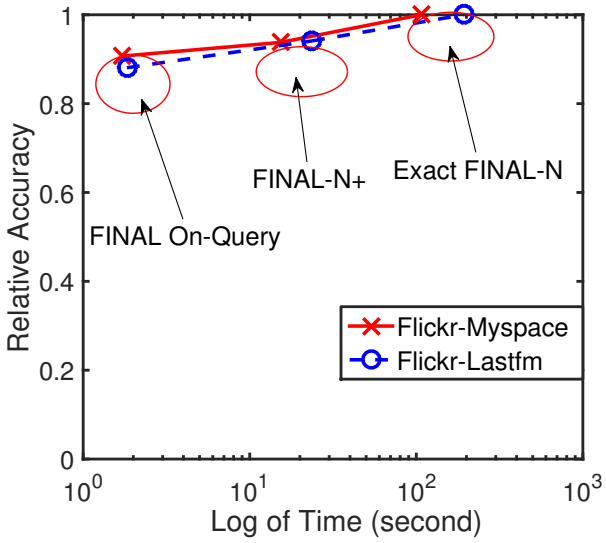


Figure 3.6: Balance of on-query alignment.

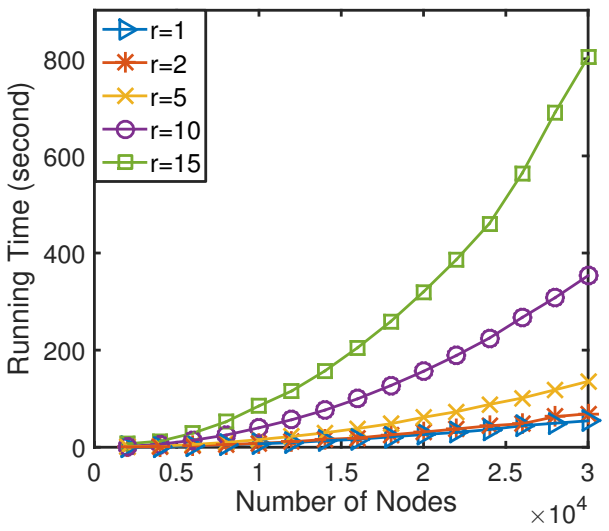


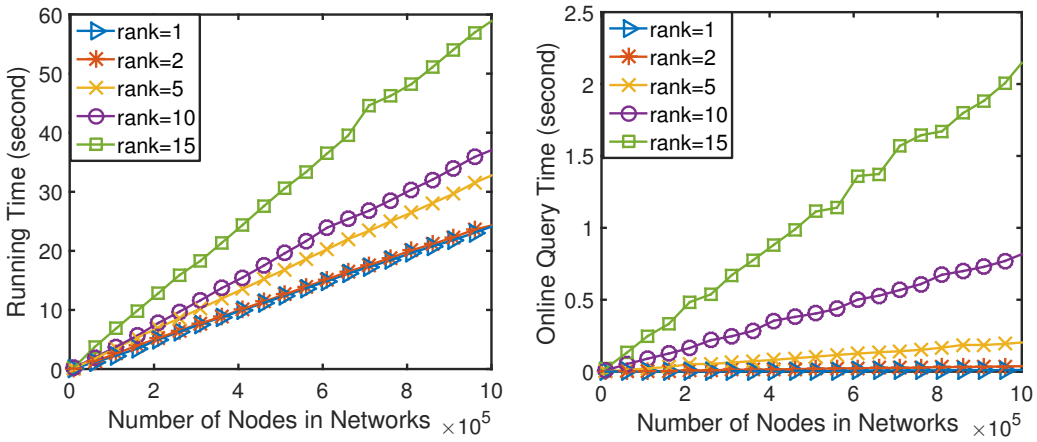
Figure 3.7: Scalability of FINAL-N+.

phase and online-query phase. As we can see from Figure 3.7, the running time is *linear* w.r.t the number of nodes in both stages, which is consistent with Lemma 3.4. In addition, the actual online-query time on the entire ArnetMiner data set (with $r = 10$) is less than 1 second, suggesting that the proposed FINAL On-Query method might be well suitable for the real-time query response.

Effectiveness and Efficiency of Moana. We only show some experimental results of the proposed method Moana, compared with other baseline methods, including (1) FINAL-P [9], (2) Umeyama's method [21], (3) HubAlign [114], (4) ModuleAlign [117], and (5) iNeat [10]. Detailed results can be referred to [11]. Figure 3.9 shows the results of Moana and baseline methods on Gr-Qc networks. Specifically, as observed in Figure 3.9 (a), the alignment accuracy of the proposed algorithm Moana is very close to its single node-level alignment counterpart FINAL-P, while outperforming other baseline methods. In addition, Figure 3.9 (b) shows that Moana obtains an up to 10× speedup compared with its single-level counterpart FINAL-P with a little loss in terms of the node-level alignment accuracy.

3.2 NON-RIGID NETWORK ALIGNMENT

Multiple networks naturally appear in many areas, ranging from social networks on various platforms, protein-protein interaction networks of different species, transaction networks at different financial institutes to the knowledge graphs constructed by different knowledge bases and the sensored networks derived from multimodal sensors (e.g., Lidar). Network



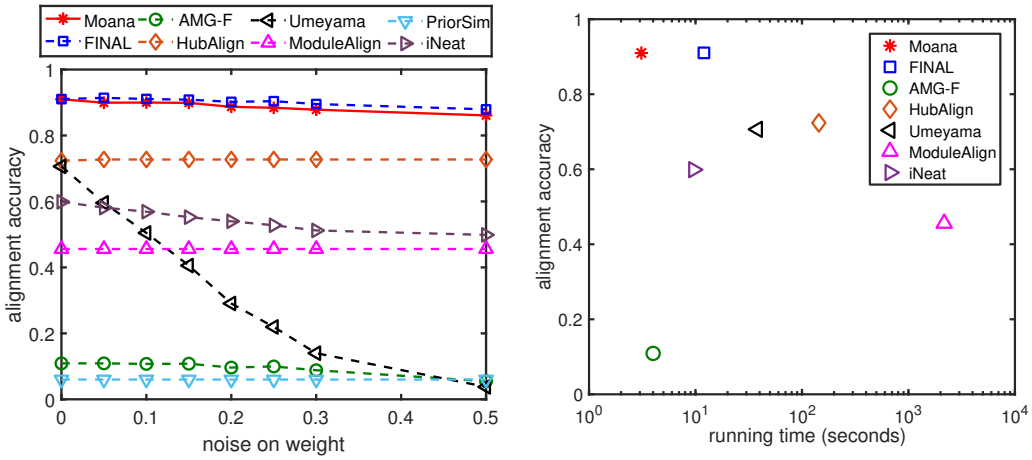
(a) Pre-compute phase.

(b) Online-query phase.

Figure 3.8: Scalability of FINAL On-Query.

alignment which aims to find the node correspondence across multiple networks is a fundamental task to integrate multiple networks into a worldview of what the input data represents. Consequently, it has drawn much attention in numerous applications. For example, by identifying the overlapping entities across multiple incomplete knowledge graphs, a unified knowledge graph can be constructed to aid knowledge completion [118]. Since many adversarial activities (e.g., smuggling) in different contexts are often covert in multiple domains, they are not quite detectable in each of the input networks alone. Network alignment can integrate these isolated networks and amplify the deceptive adversarial activities, so that they are more detectable in the composite network [4].

Despite the extensive works on network alignment, many of them explicitly or implicitly consider the alignment matrix as a linear transformation matrix. For example, many traditional graph matching based methods attempt to solve the Koopmans-Beckmann's quadratic assignment problem [119] or its relaxations, i.e., to maximize $\text{Tr}(\mathbf{S}'\mathbf{A}_1\mathbf{S}\mathbf{A}_2) + \text{Tr}(\mathbf{L}'\mathbf{S})$ where $\mathbf{S}, \mathbf{L}$ are the alignment matrix and prior cross-network node similarity matrix respectively, and $\mathbf{A}_1, \mathbf{A}_2$ are the adjacency matrices. Prior works following this path include Umeyama [21], BigAlign [18], NetAlign [5] and FINAL [9]. Since $\text{Tr}(\mathbf{S}'\mathbf{A}_1\mathbf{S}\mathbf{A}_2) = \sum_{i,j}(\mathbf{S}'\mathbf{A}_1)_{i,j}(\mathbf{A}_2\mathbf{S}')_{i,j}$, it can be alternatively viewed as first generating node feature vectors by linear transformations based on the adjacency matrices themselves (i.e., by $\mathbf{S}'\mathbf{A}_1$ and $\mathbf{A}_2\mathbf{S}'$ respectively) and then maximizing the inner product similarities between the generated feature vectors (e.g., the i-th row of $\mathbf{S}'\mathbf{A}_1$ and $\mathbf{A}_2\mathbf{S}'$). However, these methods bear some fundamental limitations, including (1) the alignment matrix $\mathbf{S}$ is leveraged as a linear transformation matrix and thus might oversimplify the complicated alignment relationships across networks; and (2) the generated feature vectors of nodes are high-dimensional and



(a) Alignment accuracy vs. edge weight noise.

(b) Balance between the accuracy and running time.

Figure 3.9: Effectiveness and efficiency of MOANA.

may fall short in their representation power. To mitigate these issues, instead of directly solving for the alignment matrix, IONE [8] and PALE [24] learn the low-dimensional node embedding vectors in different networks, based on which the alignment can be further inferred. Nevertheless, these methods ignore the node attributes that are often accompanied in real-world networks. Besides, these approaches suffer from the obstacle that node representations could be arbitrarily and imperfectly rotated and/or translated across different networks so that they might not be directly comparable.

To tackle these limitations, we go beyond the linear transformation assumption, and hypothesize that network alignment and node representation learning are mutually beneficial with each other due to the following reasons. First, *network alignment helps node representation learning*. Intuitively, if nodes are aligned across networks, the structural and attribute information of the nodes in one network can be integrated with the nodes in the other network as the auxiliary information, leading to a *far-reaching* representation learning strategy. Second, *node representation learning helps network alignment*. With the premise that node representations are of high qualities, finding node alignment in the non-Euclidean space (i.e., directly across networks) can be translated to the point set alignment problem[2] in the Euclidean space. This naturally renders the possibility of unveiling the alignment of nodes by inferring the *non-rigid* transformations which are expected to lead to the more accurate alignment.

Armed with these hypotheses, we solve the non-rigid network alignment problem by simultaneously learning node representations across multiple networks. The key ideas are

[2]We consider node representations as the point sets in the Euclidean space.

two-fold. First, to learn node representations of multiple networks, we design a new convolutional operator that can aggregate the *multi-sourced* information. Second, in order to align node representations, we design a semi-supervised multi-view non-rigid point set alignment algorithm that first learns the point set transformation function and then infers the alignment based on the transformed node representations. The main contributions are summarized as follows.

- *Problem Definition.* To our best knowledge, we are the first to address the non-rigid network alignment problem.
- *Model and Algorithms.* We develop a semi-supervised model ORIGIN which is able to simultaneously (1) learn node representations of different networks by multi-graph convolutional networks (Multi-GCN) and (2) unveil the non-rigid network alignment across multiple networks.
- *Evaluations.* Extensive experiments on real-world networks demonstrate that our proposed alignment approach (1) outperforms both the prior linear transformation based methods and node representation based methods, and (2) is efficient for node representation learning with a comparable computational time between the proposed Multi-GCN and its single-network counterpart.

3.2.1 Problem Definition

Table 3.3 summarizes the main symbols and notations used throughout the work. We use the bold uppercase letters to denote matrices (e.g., $\mathbf{X}$), bold lowercase letters (e.g., $\mathbf{x}$) for vectors and letters not in bold for scalars (e.g., α). We use $\mathbf{X}(i,j)$ to denote the entry at the intersection of the i-th row and j-th column of the matrix $\mathbf{X}$. Besides, we express $\mathbf{x}_i = \mathbf{X}(i,:)$ as the i-th row of $\mathbf{X}$ and $\mathbf{X}(:,j)$ as the j-th column of $\mathbf{X}$. We denote the transpose of matrix $\mathbf{X}$ as $\mathbf{X}'$ and the trace of matrix $\mathbf{X}$ as $\mathrm{Tr}(\mathbf{X})$.

Non-Rigid Network Alignment Problem. The concept of *non-rigid transformation* is rooted in the point set alignment problem to align the 2D or 3D point sets such that one point set can be maximally overlapped with another point set [120]. Unlike the linear or affine transformations which are restricted to some explicitly expressed transformation functions, non-rigid transformation has more flexibility to unveil the complicated alignment among point sets as it does not require any specific form of the transformation functions. Inspired by this, we translate the network alignment problem to the point set alignment

Table 3.3: Symbols and notations.

Symbols	Definitions
$\mathcal{G}_1$, $\mathcal{G}_2$	the input undirected networks
$\mathbf{A}_1, \mathbf{A}_2$	the adjacency matrices of $\mathcal{G}_1$ and $\mathcal{G}_2$
$\mathbf{X}_1, \mathbf{X}_2$	the node attributes of $\mathcal{G}_1$ and $\mathcal{G}_2$
$\mathbf{L}$	an optional $n_1 \times n_2$ prior node similarity matrix
$\mathbf{I}$	an identity matrix
$\mathbf{S}$	the output $n_1 \times n_2$ alignment matrix
$\mathbf{S}_1, \mathbf{S}_2$	sampled alignment matrices
d	the dimension of the output node representations
$\tilde{\mathbf{X}}_1, \tilde{\mathbf{X}}_2$	the node representation matrices by Inter-GCNs
$\bar{\mathbf{X}}_1, \bar{\mathbf{X}}_2$	the node representation matrices by Multi-GCN
$\mathbf{f}$	the non-rigid transformation function on $\bar{\mathbf{X}}_1$
$\mathbf{X}_1^o$	the transformed node representation matrix of $\mathcal{G}_1$ by $\mathbf{f}$
κ^1, κ^2	the kernel function for the point view and graph view
$\kappa, \mathbf{K}$	the combined kernel function and its kernel matrix
$\mathcal{L}^+$	the labeled node pairs which are aligned a priori
n_1, n_2	# of nodes in $\mathcal{G}_1, \mathcal{G}_2$
$\alpha, \alpha_1, \alpha_2, \lambda$	the regularization parameters

problem. That is, given the input networks that are known to be the non-Euclidean data [121], we aim to (1) represent the nodes in the Euclidean space so that they can be naturally viewed as point sets, and (2) align nodes in different networks (i.e., different point sets) by inferring the non-rigid transformation among them. Formally, the non-rigid network alignment problem is defined as below.

Problem 3.3. Non-Rigid Network Alignment.

Given: (1) undirected networks $\mathcal{G}_1 = \{\mathcal{V}_1, \mathbf{A}_1, \mathbf{X}_1\}$ and $\mathcal{G}_2 = \{\mathcal{V}_2, \mathbf{A}_2, \mathbf{X}_2\}$ with $\mathcal{V}_1, \mathcal{V}_2$ as the node sets where $|\mathcal{V}_1| = n_1, |\mathcal{V}_2| = n_2$, $\mathbf{A}_1, \mathbf{A}_2$ as the adjacency matrices and $\mathbf{X}_1, \mathbf{X}_2$ as the input node attribute matrices of $\mathcal{G}_1, \mathcal{G}_2$ respectively, (2) a set of labeled node pairs $\mathcal{L}^+ = \{(u_{l_i}, v_{l_i})|i = 1, \cdots, L\}$ where node u_{l_i} in $\mathcal{G}_1$ is aligned with node v_{l_i} in $\mathcal{G}_2$ a priori, (3) an optional prior cross-network node similarity matrix $\mathbf{L}$.

Find: an $n_1 \times n_2$ soft alignment matrix $\mathbf{S}$ where $\mathbf{S}(u, v)$ represents to what extent node u in $\mathcal{G}_1$ is aligned with node v in $\mathcal{G}_2$.

Remarks. If there is no prior knowledge of the cross-network node similarity matrix, we can alternatively construct $\mathbf{L}$ by some heuristics, such as node degree similarity. Though we consider network alignment problem between two input networks in this work, it is straightforward to generalize our proposed model to handle multiple network alignment.

Specifically, after computing the pair-wise alignment between each pair of networks, we can post-process (e.g., by [3]) to find the alignment among more than two input networks.

Preliminary: Graph Convolutional Networks. Graph neural networks have attracted lots of research interests in the recent years. Among others, graph convolutional networks have achieved great success in node representation learning [122, 123]. The main idea of graph convolutional networks lies in generalizing the traditional convolution operators on the Euclidean data (e.g., images) to graphs such that node information can be aggregated based on the graph structure. Here, we briefly review a spatial-based graph convolutional network (namely GraphSage [123]) that will be used as a building block in our ORIGIN model to learn node representations for a single network. Given a graph $\mathcal{G}_1$, each node-$u \in \mathcal{V}_1$ aggregates hidden representations from its neighborhood $\mathcal{N}_u$ and combines the aggregated representation with its current representation. Formally, it is formulated as

$$\tilde{\mathbf{x}}_{1,\mathcal{N}_u}^t = \text{AGGREGATE}_t(\{\tilde{\mathbf{x}}_{1,u'}^{t-1},\ \forall u' \in \mathcal{N}_u\}) \tag{3.48}$$

$$\tilde{\mathbf{x}}_{1,u}^t = \sigma\left([\tilde{\mathbf{x}}_{1,u}^{t-1} \| \tilde{\mathbf{x}}_{1,\mathcal{N}_u}^t]\mathbf{W}^t\right) \tag{3.49}$$

where $[\cdot\|\cdot]$ represents the concatenation of two vectors and $\sigma(\cdot)$ is the non-linear activation function. $\tilde{\mathbf{x}}_{1,u}^t$ is the representation of node-u in $\mathcal{G}_1$ and $\mathbf{W}^t$ denotes the weight matrix at the t-th layer. Note that when $t = 0$, $\tilde{\mathbf{x}}_{1,u}^0$ is initialized by the input attributes of node u, i.e., $\tilde{\mathbf{x}}_{1,u}^0 = \mathbf{X}_1(u,:)$. Moreover, according to [123], the GraphSage model can be instantiated by Mean, LSTM and Pooling aggregators. In this work, we choose the Mean aggregator due to its high representation power and simplicity. It is notable that GraphSage is capable of mini-batch training by uniformly sampling with replacement a fixed size of the neighboring nodes. For an input network $\mathcal{G}_1$, the unsupervised GraphSage minimizes the following loss function based on the SkipGram with negative sampling [124].

$$\mathcal{J}_{\mathcal{G}_1}(\tilde{\mathbf{X}}_1) = \sum_{u \in \mathcal{V}_1} \sum_{u' \in C_u} -\log\left(\sigma(\tilde{\mathbf{x}}'_{1,u}\tilde{\mathbf{x}}_{1,u'})\right) - N \cdot \mathbb{E}_{u'_n \sim P_n(u')} \log\left(\sigma(-\tilde{\mathbf{x}}'_{1,u}\tilde{\mathbf{x}}_{1,u'_n})\right) \tag{3.50}$$

where $u' \in C_u$ represents that node u' co-occurs with u on a fixed-length random walk, N defines the number of negative samples and $P_n(u')$ is the negative sampling distribution.

3.2.2 The ORIGIN Model

In this part, we present ORIGIN, a deep semi-supervised model that can simultaneously learn the node representations and find the non-rigid alignment across the input networks

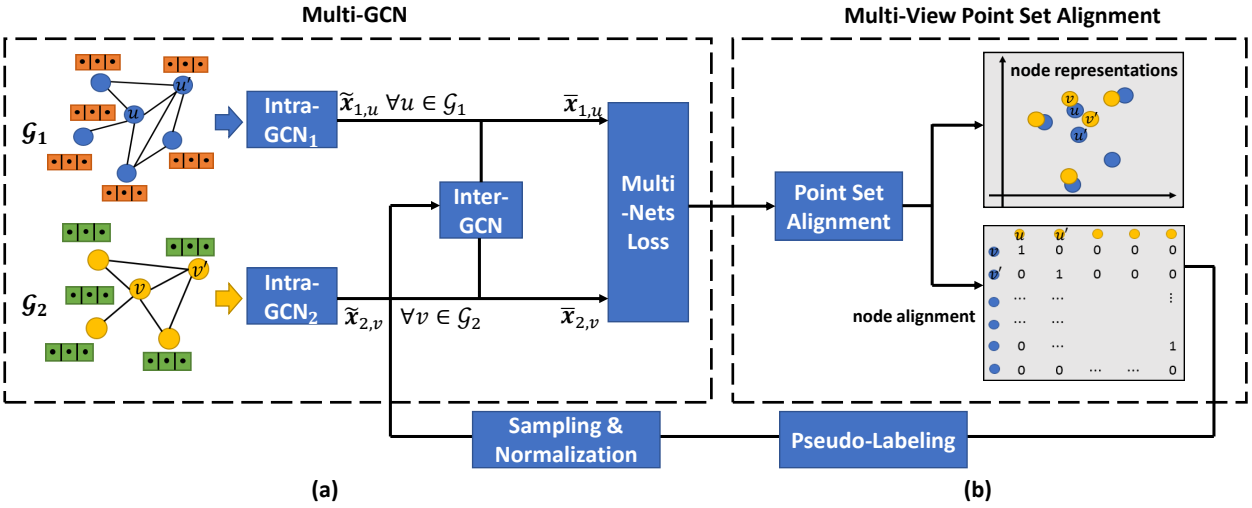


Figure 3.10: Illustration of the proposed ORIGIN model. (a) The Multi-GCN module that learns node representations of the input networks by intra-network and cross-network aggregations and combinations. (b) The point set alignment process that first displaces the node representations of $\mathcal{G}_1$ to those of $\mathcal{G}_2$, and then infers node alignment which will be fed back to Multi-GCN.

in a symbiotic way. We start by proposing a graph convolutional network model for multiple networks (Multi-GCN) to learn far-reaching node representations of the input networks. Next, we introduce a multi-view approach to unveiling the non-rigid alignment among the nodes which are represented by the point sets in the Euclidean space, followed by the optimization algorithm to effectively and efficiently learn both node representations and node alignment. The overall framework of ORIGIN is shown in Figure 3.10.

Node Representation Learning for Multiple Networks. We first present the proposed Multi-GCN model to learn node representations of multiple networks.

A - Aggregation and Combination. Many prior spatial-based graph convolutional networks (e.g., GraphSage [123]) essentially define two operators: (1) *Intra-Aggregation* that aggregates node hidden representations (or node attribute information at the first layer) from the neighboring nodes (e.g., Eq. (3.48)) underlying a single network and (2) *Intra-Combination* that combines the current hidden representation with the resultant aggregated representation of the node as the updated node representation (e.g., Eq. (3.49)). However, this might not be able to provide sufficiently informative node representations in the multiple networks scenario given that multiple networks might contain some complementary information for each other. To remedy this restrictive situation, in addition to the separate graph convolutional networks for single networks (named as Intra-GCNs), we design an *Inter-GCN* component that integrates node representations across different networks. The intuition is

to view the node alignment as the probabilistic cross-network node similarity and bridge different networks such that nodes in one network can be considered as the *virtual* neighbors of the nodes in the other network if they are likely to be aligned. For example, if node u in $\mathcal{G}_1$ is similar to node v in $\mathcal{G}_2$ (i.e., likely to be aligned), we can view node v as a *virtual* neighbor of node u. In this way, the representation of node v can be used to aggregate the neighboring node representations for node u, and vice versa. Thus, given two separate Intra-GCNs that aggregate node information in the same network and output the node representations $\tilde{\mathbf{x}}_{1,u}, \tilde{\mathbf{x}}_{2,v} \in \mathbb{R}^d,\ \forall u \in \mathcal{V}_1, v \in \mathcal{V}_2$, we define the cross-network aggregation as

$$\hat{\mathbf{x}}_{1,u} = \text{AGGREGATE}_{\text{cross}}(\tilde{\mathbf{x}}_{1,u}) = \sum_{v \in \mathcal{V}_2} \mathbf{S}(u,v)\tilde{\mathbf{x}}_{2,v} \tag{3.51}$$

$$\hat{\mathbf{x}}_{2,v} = \text{AGGREGATE}_{\text{cross}}(\tilde{\mathbf{x}}_{2,v}) = \sum_{u \in \mathcal{V}_1} \mathbf{S}(u,v)\tilde{\mathbf{x}}_{2,u}. \tag{3.52}$$

It is crucial to properly leverage the alignment matrix $\mathbf{S}$ in Eq. (3.51) and Eq. (3.52) from the following two perspectives. First (*aggregation efficiency*), the node alignment matrix is often quite dense, leading to an $O(nd)$ time complexity to compute the cross-network aggregation for each node (e.g., $\hat{\mathbf{x}}_{1,u}$) which is prohibitive especially for large-scale networks. Second (*aggregation localization*), when $\mathbf{S}$ is dense, Eq. (3.51) and Eq. (3.52) aggregate the representations of most or even all of the nodes from one network for the other, i.e., smoothing globally over the networks. This will further make the aggregated representations of different nodes less distinguishable. To overcome these issues, since we are given the labeled node pairs $\mathcal{L}^+ = \{(u_{l_i}, v_{l_i}) | i = 1, \cdots, L\}$ that indicates which nodes are aligned, we set $\mathbf{S}(u_{l_i}, v) = \mathbf{S}(u, v_{l_i}) = 0$ for all $u \in \mathcal{V}_1$ and $v \in \mathcal{V}_2$ except that $\mathbf{S}(u_{l_i}, v_{l_i}) = 1$. Besides, for all the other nodes whose alignment are unknown (e.g., $u \notin \{u_{l_i}, \forall i = 1, \cdots, L\}, v \notin \{v_{l_i}, \forall i = 1, \cdots, L\}$), we downsample the alignment matrix $\mathbf{S}$ as follows. For each node $u \notin \{u_{l_i}, \forall i = 1, \cdots, L\}$, we only preserve the K largest values $\mathbf{S}(u, v_{q_k}),\ k = 1, \cdots, K$ column-wise from $\mathbf{S}(u, :)$ for Eq. (3.51) and denote the sampled matrix as $\mathbf{S}_1$. We then normalize it such that $\sum_{k=1}^{K} \mathbf{S}_1(u, v_{q_k}) = 1$. Similarly, we sample K values $\mathbf{S}(u_{p_k}, v)$ for each node $v \notin \{v_{l_i}, \forall i = 1, \cdots, L\}$ row-wise from $\mathbf{S}(:, v)$ and denote it as $\mathbf{S}_2$ where $\sum_{k=1}^{K} \mathbf{S}_2(u_{p_k}, v) = 1$ after normalization. This sampling and normalization process is summarized in Algorithm 3.5. The cross-network aggregation is re-written as

$$\begin{aligned} \hat{\mathbf{x}}_{1,u} &= \text{AGGREGATE}_{\text{cross}}(\tilde{\mathbf{x}}_{1,u}) = \sum_{k=1}^{K} \mathbf{S}_1(u, v_{q_k})\tilde{\mathbf{x}}_{2,v_{q_k}} \\ \hat{\mathbf{x}}_{2,v} &= \text{AGGREGATE}_{\text{cross}}(\tilde{\mathbf{x}}_{2,v}) = \sum_{k=1}^{K} \mathbf{S}_2(u_{p_k}, v)\tilde{\mathbf{x}}_{1,u_{p_k}} \end{aligned} \tag{3.53}$$

Algorithm 3.5: SAMPLE($\mathbf{S}, K$).

Input : (1) the current alignment matrix $\mathbf{S}$, (2) the labeled node pairs $\mathcal{L}^+$ and (3) the sample size K.
Output: sampled alignment matrix $\mathbf{S}_1, \mathbf{S}_2$.
Initialize $\mathbf{S}_1 = \mathbf{S}_2 = \mathbf{S}$;
for $i = 1 \to L$ **do**
  Set $\mathbf{S}_1(u_{l_i}, v_{l_i}) = \mathbf{S}_2(u_{l_i}, v_{l_i}) = 1$;
  Set $\mathbf{S}_1(u, v_{l_i}) = \mathbf{S}_2(u_{l_i}, v) = 0,\ \forall u \neq u_{l_i}, v \neq v_{l_i}$;
end
Preserve top-K nonzero elements in $\mathbf{S}_1(u, :),\ \forall u \notin \{u_{l_i}, \forall i = 1, \cdots, L\}$;
Preserve top-K nonzero elements in $\mathbf{S}_2(:, v),\ \forall v \notin \{v_{l_i}, \forall i = 1, \cdots, L\}$;
Normalize $\mathbf{S}_1(u, :),\ \forall u = 1, \cdots, n_1$;
Normalize $\mathbf{S}_2(:, v),\ \forall v = 1, \cdots, n_2$;
Output $\mathbf{S}_1,\ \mathbf{S}_2$.

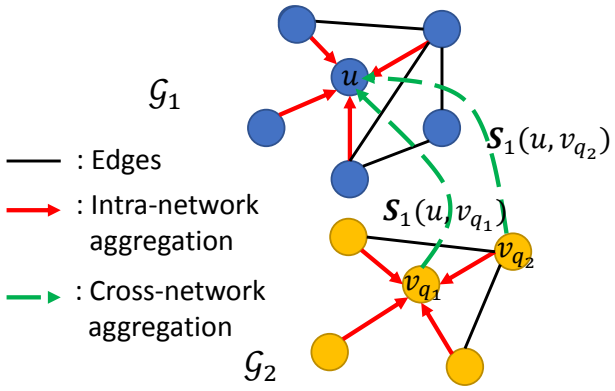


Figure 3.11: An illustrative example of the aggregations in the multi-GCN model ($K = 2$). Red solid lines represent the aggregations from node neighborhood within a single network in Intra-GCN while green dashed lines represent the cross-network aggregations in Inter-GCN.

We show an illustrative example in Figure 3.11 where $K = 2$ and nodes v_{q_1}, v_{q_2} are sampled for cross-network aggregation for $\tilde{\mathbf{x}}_u$ through $\mathbf{S}_1(u, v_{q_1})$ and $\mathbf{S}_1(u, v_{q_2})$ respectively. For cross-network combination, we use

$$\begin{aligned} \bar{\mathbf{x}}_{1,u} &= \text{COMBINE}_{\text{cross}}(\tilde{\mathbf{x}}_{1,u}, \hat{\mathbf{x}}_{1,u}) = [\tilde{\mathbf{x}}_{1,u} \| \hat{\mathbf{x}}_{1,u}]\mathbf{W}_{cross} + \mathbf{b}_1 \\ \bar{\mathbf{x}}_{2,v} &= \text{COMBINE}_{\text{cross}}(\tilde{\mathbf{x}}_{2,v}, \hat{\mathbf{x}}_{2,v}) = [\tilde{\mathbf{x}}_{2,v} \| \hat{\mathbf{x}}_{2,v}]\mathbf{W}_{cross} + \mathbf{b}_2 \end{aligned} \tag{3.54}$$

where $\bar{\mathbf{x}}_{1,u}, \bar{\mathbf{x}}_{2,v} \in \mathbb{R}^d$ are the output node representations by the proposed Multi-GCN model. The weight matrix $\mathbf{W}_{\text{cross}} \in \mathbb{R}^{2d \times d}$ is shared in both equations as it basically measures how to combine the representations learned by Intra-GCNs and by Inter-GCN for cross-network combinations.

B - Loss Functions. To learn the far-reaching node representations that can simultaneously maintain the local structural information within a single network and the cross-network representation consistency, we aim to minimize the loss function

$$\mathcal{J}_{GCN} = \mathcal{J}_{\mathcal{G}_1}(\bar{\mathbf{X}}_1) + \mathcal{J}_{\mathcal{G}_2}(\bar{\mathbf{X}}_2) + \lambda \mathcal{J}_{\text{cross}}(\bar{\mathbf{X}}_1, \bar{\mathbf{X}}_2) \tag{3.55}$$

where $\mathcal{J}_{\mathcal{G}_1}(\bar{\mathbf{X}}_1), \mathcal{J}_{\mathcal{G}_2}(\bar{\mathbf{X}}_2)$ have the same formula as Eq. (3.50), but are minimized over $\bar{\mathbf{X}}_1, \bar{\mathbf{X}}_2$ respectively instead of $\tilde{\mathbf{X}}_1, \tilde{\mathbf{X}}_2$. Minimizing $\mathcal{J}_{\mathcal{G}_1}(\bar{\mathbf{X}}_1)$, $\mathcal{J}_{\mathcal{G}_2}(\bar{\mathbf{X}}_2)$ encourages the representations of the nearby nodes to be similar and the representations of disparate nodes to be dissimilar. Besides, to impose the consistency of the node representations between different networks, we minimize the distance between the node representations of one network and those computed by the aggregations from the other network via alignment. Specifically, given node representations $\bar{\mathbf{x}}_{1,u}$ and $\bar{\mathbf{x}}_{2,v}$, we aim to minimize the following disagreement loss.

$$\mathcal{J}_{\text{cross}}(\bar{\mathbf{X}}_1, \bar{\mathbf{X}}_2) = \sum_{u \in \mathcal{V}_1} \|\bar{\mathbf{x}}_{1,u} - \sum_{k=1}^{K} \mathbf{S}_1(u, v_{q_k}) \bar{\mathbf{x}}_{2,v_{q_k}}\|_2^2 + \sum_{v \in \mathcal{V}_2} \|\bar{\mathbf{x}}_{2,v} - \sum_{k=1}^{K} \mathbf{S}_2(u_{p_k}, v) \bar{\mathbf{x}}_{1,u_{p_k}}\|_2^2 \tag{3.56}$$

Note that for $(u_{l_i}, v_{l_i}) \in \mathcal{L}^+$, since $\mathbf{S}_1(u_{l_i}, v_{l_i}) = 1$ is the only nonzero entry in the row of u_{l_i}, the first term in Eq. (3.56) is equivalent to $\|\bar{\mathbf{x}}_{1,u_{l_i}} - \bar{\mathbf{x}}_{2,v_{l_i}}\|_2^2$, enforcing the representation of node u_{l_i} in $\mathcal{G}_1$ to be close to that of its aligned node v_{l_i} in $\mathcal{G}_2$. However, one computational issue that resides in the Eq. (3.56) is that the node representations of all the other unlabeled nodes are nested with each other. For example, the first term needs the node representations $\bar{\mathbf{x}}_{2,v_{q_k}}$, $\forall k = 1, \cdots, K$ which requires to explicitly calculate all node representations of $\mathcal{G}_2$ (i.e., $\bar{\mathbf{X}}_2$) in each iteration in the worst case. This is impractical especially when one wants to use stochastic training methods. Thus, we further simplify the terms in Eq. (3.56). For brevity, we only take $\sum_{k=1}^{K} \mathbf{S}_1(u, v_{q_k}) \bar{\mathbf{x}}_{2,v_{q_k}}$ as an example, and rewrite it as below.

$$\begin{aligned}
\sum_{k=1}^{K} \mathbf{S}_1(u, v_{q_k}) \bar{\mathbf{x}}_{2,v_{q_k}} &= \sum_{k=1}^{K} \mathbf{S}_1(u, v_{q_k}) \left([\tilde{\mathbf{x}}_{2,v_{q_k}} \| \hat{\mathbf{x}}_{2,v_{q_k}}] \mathbf{W}_{\text{cross}} + \mathbf{b}_2 \right) \\
&= \sum_{k=1}^{K} \left[\sum_{k'=1}^{K} \mathbf{S}_1(u, v_{q_k}) \mathbf{S}_2(u_{p_{k'}}, v_{q_k}) \tilde{\mathbf{x}}_{1,u_{p_{k'}}} \mathbf{W}_{c_2} + \mathbf{S}_1(u, v_{q_k}) \left(\tilde{\mathbf{x}}_{2,v_{q_k}} \mathbf{W}_{c_1} + \mathbf{b}_2 \right) \right] \\
&= \sum_{k=1}^{K} \mathbf{S}_1(u, v_{q_k}) (\tilde{\mathbf{x}}_{2,v_{q_k}} \mathbf{W}_{c_1} + \mathbf{b}_2) + \sum_{k'=1} (\mathbf{S}_1 \mathbf{S}_2')(u, u_{p_{k'}}) \tilde{\mathbf{x}}_{1,u_{p_{k'}}} \mathbf{W}_{c_2} \\
&= \left[\sum_{k=1}^{K} \mathbf{S}_1(u, v_{q_k}) \tilde{\mathbf{x}}_{2,v_{q_k}} \| \sum_{k'=1} (\mathbf{S}_1 \mathbf{S}_2')(u, u_{p_{k'}}) \tilde{\mathbf{x}}_{1,u_{p_{k'}}} \right] \mathbf{W}_{\text{cross}} + \mathbf{b}_2
\end{aligned} \tag{3.57}$$

where $\mathbf{W}_{c_1}$ and $\mathbf{W}_{c_2}$ are the first and second d rows of $\mathbf{W}_{\text{cross}}$ respectively. Based on the above equations, we can rethink of the computation for the aggregated representation of $\bar{\mathbf{x}}_{1,u}$ (i.e., $\sum_{k=1}^{K} \mathbf{S}_1(u, v_{q_k})\bar{\mathbf{x}}_{2,v_{q_k}}$) as two steps. First, we concatenate (1) the result of the cross-network aggregation $\hat{\mathbf{x}}_{1,u}$ and (2) the result (denoted by $\bar{\mathbf{y}}_{1,u}$) of the aggregation among the nodes $u_{p_{k'}}$ from the same network $\mathcal{G}_1$ weighted by $(\mathbf{S}_1\mathbf{S}_2')(u, u_{p_{k'}})$. Then, it is equivalent to

$$\sum_{k=1}^{K} \mathbf{S}_1(u, v_{q_k})\bar{\mathbf{x}}_{2,v_{q_k}} = [\hat{\mathbf{x}}_{1,u} \| \bar{\mathbf{y}}_{1,u}]\mathbf{W}_{\text{cross}} + \mathbf{b}_2 \tag{3.58}$$

Nevertheless, as $\mathbf{S}_1\mathbf{S}_2'$ is likely to be a dense matrix, the aggregations for $\bar{\mathbf{y}}_{1,u}$ gather the information globally from most of the other nodes, leading to a less emblematic $\bar{\mathbf{y}}_{1,u}$ and a higher computational cost. To address this issue, we only preserve the nonzero $(\mathbf{S}_1\mathbf{S}_2')(u, u_{p_{k'}})$, $\forall u_{p_{k'}} \in C_u$. In this way, the extra node representations that are needed only include $\tilde{\mathbf{x}}_{1,u_{p_{k'}}}$ which can be efficiently calculated by feeding node $u_{p_{k'}} \in C_u$ to the Intra-GCN$_1$ module. Accordingly, we can simplify the second term in Eq. (3.56) similarly.

Multi-View Point Set Alignment. After we obtain the node representations learned by the proposed Multi-GCN model, it seems that we can simply learn whether node u in $\mathcal{G}_1$ is aligned with node v in $\mathcal{G}_2$ based on their node representations $\bar{\mathbf{x}}_{1,u}$ and $\bar{\mathbf{x}}_{2,v}$. However, as the Multi-GCN model only preserves the structural consistency within the same networks (i.e., Eq. (3.50)) and the consistency between node representations and their aggregated representations from the other network (i.e., Eq. (3.56)), node representations $\bar{\mathbf{x}}_{1,u}$ and $\bar{\mathbf{x}}_{2,v}$ are still likely not to be close with each other in the Euclidean space even if node u and node v are supposed to be aligned. This limitation could result in a sub-optimal alignment or even totally mislead the alignment. To mitigate this limitation, we translate the network alignment problem over the nodes to a non-rigid point set alignment (PSA) problem where each point is represented by the representation of the corresponding node in the Euclidean space. Specifically, given a set of labeled node alignment $\mathcal{L}^+ = \{(u_{l_i}, v_{l_i}) | i = 1, \cdots, L\}$, we want to displace each point $\bar{\mathbf{x}}_{1,u_{l_i}}$ towards its aligned point $\bar{\mathbf{x}}_{2,v_{l_i}}$ by some non-rigid vector-valued transformation function $\mathbf{f} \in \mathbb{R}^d$ such that the maximum point-to-point overlaps can be achieved. Mathematically, this can be formulated as a functional minimization problem.

$$\min_{\mathbf{f}} \quad \sum_{i=1}^{L} \|\bar{\mathbf{x}}_{1,u_{l_i}} + \frac{1}{2}\mathbf{f}(\bar{\mathbf{x}}_{1,u_{l_i}}) - \bar{\mathbf{x}}_{2,v_{l_i}}\|_2^2 \tag{3.59}$$

However, Eq. (3.59) is an ill-posed problem without any constraints imposed on $\mathbf{f}$. Instead, we model the above optimization problem by requiring the non-rigid function $\mathbf{f}$ to lie within

a specific functional space, namely a reproducing kernel Hilbert space (RKHS) denoted by $\mathcal{H}$. Then we have

$$\min_{\mathbf{f}} \quad \sum_{i=1}^{L} \|\bar{\mathbf{x}}_{1,u_{l_i}} + \frac{1}{2}\mathbf{f}(\bar{\mathbf{x}}_{1,u_{l_i}}) - \bar{\mathbf{x}}_{2,v_{l_i}}\|_2^2 + \alpha\|\mathbf{f}\|_{\mathcal{H}}^2 \tag{3.60}$$

where $\|\mathbf{f}\|_{\mathcal{H}}^2$ is the RKHS norm of $\mathbf{f}$ in $\mathcal{H}$ and α is the regularization parameter. In addition, since each point (e.g., $\bar{\mathbf{x}}_{1,u_{l_i}}$) intrinsically has two interpretations (or views): points in the Euclidean space (i.e., node representations) and the corresponding nodes of the networks in the non-Euclidean graph space, we further consider to divide $\mathcal{H}$ into two RKHS $\mathcal{H}^1, \mathcal{H}^2$ by $\mathcal{H} = \mathcal{H}^1 \oplus \mathcal{H}^2$ such that

$$\mathcal{H} = \left\{\mathbf{f} | \mathbf{f}(\mathbf{x}) = \mathbf{f}^1(\mathbf{x}) + \mathbf{f}^2(\mathbf{x}), \mathbf{f}^1 \in \mathcal{H}^1, \mathbf{f}^2 \in \mathcal{H}^2\right\} \tag{3.61}$$

and the RKHS norm $\|\mathbf{f}\|_{\mathcal{H}}^2$ can be re-written as

$$\|\mathbf{f}\|_{\mathcal{H}}^2 = \min_{\substack{\mathbf{f}=\mathbf{f}^1+\mathbf{f}^2 \\ \mathbf{f}^1\in\mathcal{H}^1 \\ \mathbf{f}^2\in\mathcal{H}^2}} \alpha_1\|\mathbf{f}^1\|_{\mathcal{H}^1}^2 + \alpha_2\|\mathbf{f}^2\|_{\mathcal{H}^2}^2 + \mu\sum_{j=1}^{n_1-L}\left[\mathbf{f}^1(\bar{\mathbf{x}}_{1,u_{r_j}}) - \mathbf{f}^2(\bar{\mathbf{x}}_{1,u_{r_j}})\right]^2 \tag{3.62}$$

where $\mathbf{f}^1(\bar{\mathbf{x}}_1)$, $\mathbf{f}^2(\bar{\mathbf{x}}_1)$ are two transformation functions in the RKHS $\mathcal{H}^1, \mathcal{H}^2$ corresponding to the point view and graph view, respectively. Besides, $\mathcal{U} = \{u_{r_j} | j = 1, \cdots, n_1 - L\}$ represents the unlabeled nodes in $\mathcal{G}_1$ whose alignment with the nodes in $\mathcal{G}_2$ are not labeled. Intuitively, the term $\left[\mathbf{f}^1(\bar{\mathbf{x}}_{1,u_{r_j}}) - \mathbf{f}^2(\bar{\mathbf{x}}_{1,u_{r_j}})\right]^2$ regularizes the transformation functions $\mathbf{f}^1, \mathbf{f}^2$ over the unlabeled node u_{r_j} to be consistent in two different views (i.e., moving $\bar{\mathbf{x}}_{1,u_{r_j}}$ coherently in two views). According to [125], let $\mathcal{H}^1, \mathcal{H}^2$ be with the reproducing kernels κ^1, κ^2, then the RKHS $\mathcal{H}$ is with the reproducing kernel

$$\kappa(u, u') = \phi(u, u') - \mu\mathbf{a}_u\mathbf{\Psi}\mathbf{a}'_{u'}. \tag{3.63}$$

Here, $\phi(u, u') = \alpha_1^{-1}\kappa^1(u, u') + \alpha_2^{-1}\kappa^2(u, u')$ and $\mathbf{a}_u = \alpha_1^{-1}\mathbf{k}^1_{u\mathcal{U}} - \alpha_2^{-1}\mathbf{k}^2_{u\mathcal{U}}$ where $\mathbf{k}^s_{u\mathcal{U}} = [\kappa^s(u, u_{r_j}), u_{r_j} \in \mathcal{U}]$, $\forall s = 1, 2$ is a row vector measuring the kernel values between u and all the other unlabeled nodes in $\mathcal{U}$. Besides, $\mathbf{\Psi}$ is a positive definite matrix computed by $\mathbf{\Psi} = (\mathbf{I} + \mu\mathbf{\Phi})^{-1}$ where $\mathbf{\Phi}$ denotes the kernel matrix of $\phi(u, u')$ over all the unlabeled nodes. Furthermore, by the Representer Theorem [126], the solution to Eq. (3.62) is a function that

$$\mathbf{f}(\bar{\mathbf{x}}_{1,u}) = \mathbf{K}(u, \mathcal{I})\mathbf{\Gamma} \tag{3.64}$$

where $\mathcal{I} = \{u_{l_i} | i = 1, \cdots, L\}$ includes the indices of the labeled nodes in $\mathcal{G}_1$, $\mathbf{K}$ represents the kernel matrix corresponding to the kernel κ and $\mathbf{\Gamma}$ includes the coefficients to be solved. By substituting Eq. (3.64) to Eq. (3.60), we have

$$\min_{\mathbf{\Gamma}} \quad \mathcal{J}_{\text{PSA}} = \sum_{i=1}^{L} \|\mathbf{x}_{u_{l_i}} + \frac{1}{2}\mathbf{K}(u_{l_i}, \mathcal{I})\mathbf{\Gamma} - \mathbf{y}_{v_{l_i}}\|_2^2 + \alpha \text{Tr}(\mathbf{\Gamma}'\mathbf{K}_{\mathcal{I}}\mathbf{\Gamma}) \tag{3.65}$$

where $\mathbf{K}_{\mathcal{I}} = \mathbf{K}(\mathcal{I}, \mathcal{I})$.

To construct the kernel κ based on Eq. (3.63), we use the Gaussian RBF kernel $\kappa^1(u, u') = \exp(-\|\bar{\mathbf{x}}_{1,u} - \bar{\mathbf{x}}_{1,u'}\|_2^2)$ for the point view. In terms of the graph view, many graph kernels have been proposed, such as diffusion kernel [127], p-step random walk kernel [128]. In this work, we choose the 1-step random walk kernel due to its computational simplicity. In particular, the kernel matrix corresponding to the kernel κ^2 is formulated as $\mathbf{K}^2 = 2\mathbf{I} - \tilde{\mathbf{L}} = \mathbf{I} + \mathbf{D}_1^{-\frac{1}{2}}\mathbf{A}_1\mathbf{D}_1^{-\frac{1}{2}}$ where $\mathbf{D}_1$ is the diagonal degree matrix of $\mathbf{A}_1$. By optimizing Eq. (3.65), we can solve for $\mathbf{\Gamma}$ and compute the transformed node representations of $\mathcal{G}_1$ by $\mathbf{x}^o_{1,u} = \bar{\mathbf{x}}_{1,u} + \frac{1}{2}\mathbf{K}(u, \mathcal{I})\mathbf{\Gamma}$. Such transformed node representations will then be compared with $\bar{\mathbf{X}}_2$ to infer the alignment with the nodes in $\mathcal{G}_2$. Specifically, we calculate the cross-network node similarity between node u in $\mathcal{G}_1$ and node v in $\mathcal{G}_2$ as the alignment matrix by $\mathbf{S}(u, v) = \exp(-\|\mathbf{x}^0_{1,u} - \mathbf{x}^o_{2,v}\|_2^2)$ where $\mathbf{x}^o_{2,v} = \bar{\mathbf{x}}_{2,v}$.

Optimization Algorithm. The overall loss function of the proposed model is

$$\mathcal{J} = \underbrace{\mathcal{J}_{\mathcal{G}_1}(\bar{\mathbf{X}}_1) + \mathcal{J}_{\mathcal{G}_2}(\bar{\mathbf{X}}_2) + \lambda \mathcal{J}_{\text{cross}}(\bar{\mathbf{X}}_1, \bar{\mathbf{X}}_2)}_{\text{Multi-GCN}} + \underbrace{\mathcal{J}_{\text{PSA}}}_{\text{point set alignment}} \tag{3.66}$$

To minimize the above loss function, it is straightforward to simultaneously learn all the parameters in an alternating manner. However, the main drawbacks of this approach include: (1) if the node representations of $\mathcal{G}_1$ and $\mathcal{G}_2$ are not representative enough, the inferred alignment could be suboptimal or even misleading, and (2) at the initial stages, as the alignment matrix $\mathbf{S}$ is expected to be imprecise, the learning of node representations is very likely to be misled (e.g., in Eq. (3.53)).

Instead, we develop the optimization algorithm, where each training cycle is divided into two stages. In the first stage, we train the proposed Multi-GCN model to learn node representations with the current alignment matrix $\mathbf{S}$. In this work, we use GraphSage model with Mean aggregators [123] for Intra-GCN$_1$ and Intra-GCN$_2$ and hence the Multi-GCN model can be optimized by mini-batched stochastic gradient descent (SGD). We note that other GCN models can also be also used as the Intra-GCNs (e.g., [129]). In the second stage,

Algorithm 3.6: Non-rigid Network Alignment (ORIGIN).

Input : (1) undirected networks $\mathcal{G}_1 = \{\mathcal{V}_1, \mathbf{A}_1, \mathbf{X}_1\}$ and $\mathcal{G}_2 = \{\mathcal{V}_2, \mathbf{A}_2, \mathbf{X}_2\}$, (2) a set of labeled cross-network node pairs $\mathcal{L}^+$ that are aligned, (3) the prior node similarity matrix $\mathbf{L}$, (4) the parameters $\alpha, \alpha_1, \alpha_2, \lambda, K, \rho$, (5) the total number of iterations $\text{iter}_{\max}$.

Output: (1) the alignment matrix $\mathbf{S}$ between $\mathcal{G}_1, \mathcal{G}_2$.

Initialize the alignment matrix $\mathbf{S}$ by $\mathbf{L}$;

Compute the kernel matrix $\mathbf{K}^2 = \mathbf{I} + \mathbf{D}_1^{-\frac{1}{2}}\mathbf{A}_1\mathbf{D}_1^{-\frac{1}{2}}$;

for $iter = 1 \to iter_{max}$ **do**

  Compute $\mathbf{S}_1, \mathbf{S}_2$ by SAMPLE($\mathbf{S}$, K);

  Generate mini-batches $\mathcal{B}^1 = \{\mathcal{B}_1^1, \cdots, \mathcal{B}_B^1\}, \mathcal{B}^2 = \{\mathcal{B}_1^2, \cdots, \mathcal{B}_B^2\}$ for $\mathcal{G}_1, \mathcal{G}_2$ by GraphSage models;

  for $b = 1 \to B$ **do**

    Generate $\tilde{\mathbf{x}}_{1,u}, \tilde{\mathbf{x}}_{2,v}$ for $u \in \mathcal{B}_b^1, v \in \mathcal{B}_b^2$ by GraphSage;

    Compute $\hat{\mathbf{x}}_{1,u}, \hat{\mathbf{x}}_{2,v}$ for $u \in \mathcal{B}_b^1, v \in \mathcal{B}_b^2$ by Eq. (3.53);

    Compute $\bar{\mathbf{x}}_{1,u}, \bar{\mathbf{x}}_{2,v}$ for $u \in \mathcal{B}_b^1, v \in \mathcal{B}_b^2$ by Eq. (3.54);

    Update hidden layer parameters by optimizing $\mathcal{J}_{\text{GCN}}$;

  end

  Compute $\bar{\mathbf{X}}_1, \bar{\mathbf{X}}_2$ by hidden layer parameters;

  Compute $\mathbf{K}^1$ by $\mathbf{K}^1(u, u') = \exp(-\|\bar{\mathbf{x}}_{1,u} - \bar{\mathbf{x}}_{1,u'}\|_2^2)$;

  Compute $\mathbf{K}$ by Eq. (3.63);

  Compute $\mathbf{\Gamma}$ by Eq. (3.68);

  Compute $\mathbf{X}_1^o = \bar{\mathbf{X}}_1 + \frac{1}{2}\mathbf{K}(:, \mathcal{I})\mathbf{\Gamma}$ for $\mathcal{G}_1$ and $\mathbf{X}_2^o = \bar{\mathbf{X}}_2$;

  Update alignment $\mathbf{S}$ by $\mathbf{S}(u, v) = \exp(-\|\mathbf{x}_{1,u}^o - \mathbf{x}_{2,v}^o\|_2^2)$;

  Generate pseudo labels based on $\mathbf{S}$;

  Anneal $K \leftarrow K/\rho$;

end

Output the alignment matrix $\mathbf{S}$.

we learn the parameter $\mathbf{\Gamma}$ by solving the optimization problem Eq. (3.65). Specifically, we compute the gradient of Eq. (3.65) w.r.t. $\mathbf{\Gamma}$ as

$$\frac{\partial \mathcal{J}_{\text{PSA}}}{\partial \mathbf{\Gamma}} = \frac{1}{2}\mathbf{K}'_{\mathcal{I}}\mathbf{K}_{\mathcal{I}}\mathbf{\Gamma} + \mathbf{K}'_{\mathcal{I}}(\bar{\mathbf{X}}_1(\mathcal{I}, :) - \bar{\mathbf{X}}_2(\mathcal{I}, :)) + 2\alpha\mathbf{K}_{\mathcal{I}}\mathbf{\Gamma} \tag{3.67}$$

Then we can obtain the solution of $\mathbf{\Gamma}$ by gradient descent.

$$\mathbf{\Gamma} = \mathbf{\Gamma} - \eta\frac{\partial \mathcal{J}_{\text{PSA}}}{\partial \mathbf{\Gamma}} \tag{3.68}$$

where η is the learning rate. Note that the time complexity of Eq. (3.68) in each iteration is $O(|\mathcal{L}^+|^2 d)$ which is sub-quadratic w.r.t. the number of nodes n. After that, we calculate

the displaced node representations of $\mathcal{G}_1$ by $\mathbf{X}_1^o = \bar{\mathbf{X}}_1 + \frac{1}{2}\mathbf{K}(:, \mathcal{I})\mathbf{\Gamma}$, followed by computing the cross-network node similarity matrix as the alignment matrix $\mathbf{S}$. Next, the alignment matrix $\mathbf{S}$ will be fed back to Multi-GCN after the pseudo-labeling and sampling steps. To generate the pseudo-labels indicating which node in $\mathcal{G}_1$ is aligned with which node in $\mathcal{G}_2$, we first conduct a greedy matching process on $\mathbf{S}$ to obtain all one-to-one node alignment [3], and then leverage the alignment consistency proposed in [9] to select the confident alignment as the pseudo labels. To be specific, note that the alignment consistency assumes if two nodes are aligned, their corresponding close neighbors are likely to be aligned. In this way, we heuristically select the alignment between node u and node v obtained by greedy matching as the pseudo alignment label, if nodes (u, v) are the respective neighbors of nodes $(u_{l_i}, v_{l_i}) \in \mathcal{L}^+$. The overall optimization algorithm is summarized in Algorithm 3.6. Note that as the model is trained, matrix $\mathbf{S}$ should indicate more accurate node alignment. For this reason, we reduce the sample size K used in Algorithm 3.5 by ρ (Line 3.6).

3.2.3 Experimental Evaluations

In this part, we present experimental results of the proposed model ORIGIN. We evaluate in the following aspects:

- *Effectiveness*: How accurate is our algorithm to align networks and how robust is our algorithm to the parameters?
- *Efficiency*: How fast is our algorithm?

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

Datasets. We evaluate the proposed model on four real-world networks. The statistics of all datasets are summarized in Table 3.4.

- *Citation networks*: We consider two citation networks: Cora and Citeseer[3]. Each node represents a document and each edge indicates a citation link between two documents. We use the bag-of-words representations of the documents as the node attributes. For both networks, we convert the originally directional edges to undirected to make the networks undirected.
- *Social networks*: We consider two social networks including Foursquare and Twitter [130]. Each node in the networks represents a user and each edge represents the friendship between two users. For each node in the networks, we compute the degree, the

[3] `https://linqs.soe.ucsc.edu/data`

Table 3.4: Data statistics.

Category	Network	# of Nodes	# of Edges	# of Attributes
Citation	Cora	2,708	5,429	1,433
Citation	Citeseer	3,327	4,732	3,703
Social	Foursquare	5,313	54,233	5
Social	Twitter	5,120	130,575	5

number of edges in the egonet, PageRank score, betweenness and closeness centralities and use them as node attributes. We then convert them to undirected networks.

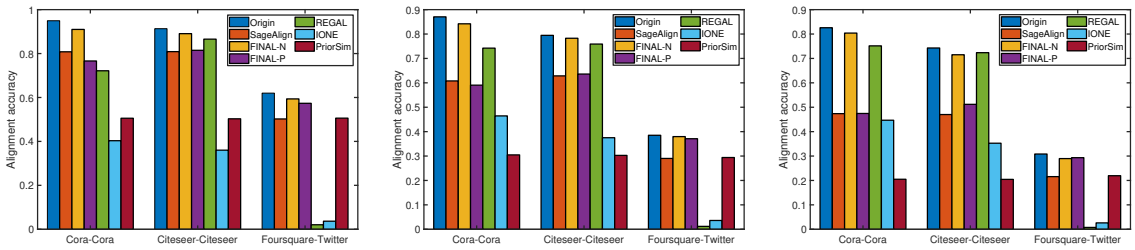
We build the following three scenarios to evaluate the alignment performance. In each scenario, we randomly select 50%, 30% and 20% cross-network node pairs from the ground-truth as labeled alignment $\mathcal{L}^+$.

- *Cora-1 vs. Cora-2.* Given the Cora network (denoted by $\mathcal{G}_1 = \{\mathcal{V}_1, \mathbf{A}_1, \mathbf{X}_1\}$), we first generate a random permutation matrix $\mathbf{P}$ which is used as the ground-truth alignment. We treat the permutated matrix $\mathbf{A}_2 = \mathbf{P}'\mathbf{A}_1\mathbf{P}$ and $\mathbf{X}_2 = \mathbf{P}'\mathbf{X}_1$ as the adjacency matrix and node attributes of the second network $\mathcal{G}_2$. We also randomly remove 5%, 15% edges from two networks and add 10%, 15% noise on the node attributes of two networks, respectively. We compute node degree similarity for $\mathbf{L}$.

- *Citeseer-1 vs. Citeseer-2.* This scenario is built similarly.

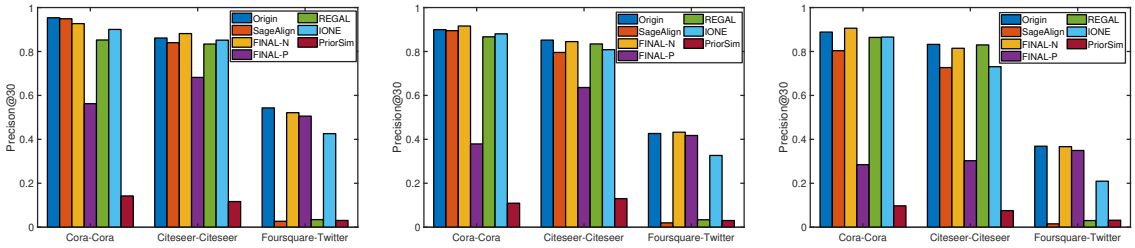
- *Foursquare vs. Twitter.* In this scenario, we aim to align nodes in Foursquare and Twitter networks. There are 1,609 common nodes between two networks which are used as the ground-truth to evaluate the alignment. Besides, we compute the degree similarity matrix as $\mathbf{L}$.

Baseline Methods. We compare our method ORIGIN with the following network alignment algorithms, including (1) SageAlign that learns node representations by two separate GraphSage models [123], followed by the proposed point set alignment algorithm, (2) FINAL-N [9, 12], (3) FINAL-P which is a non-attributed variant of FINAL-N, (4) REGAL [29], (5) IONE [8], and (6) PriorSim which aligns the nodes directly based on the prior node similarity matrix $\mathbf{L}$. To make a fair comparisons, we set $\mathbf{L}(u_{l_i}, :) = \mathbf{L}(:, v_{l_i}) = 0$ except that $\mathbf{L}(u_{l_i}, v_{l_i}) = 1,\ \forall i = 1, \cdots, L$ for FINAL-N, FINAL-P and PriorSim to incorporate the label information. For REGAL, we slightly modify the original released code by setting the constructed node similarity matrix to be factorized by $\mathbf{L}_{\text{REGAL}}(u_{l_i}, v_{l_i}) = 1,\ \forall i = 1, \cdots, L$.



(a) 50% labeled alignment. (b) 30% labeled alignment. (c) 20% labeled alignment.

Figure 3.12: Alignment accuracy with different amount of labeled alignment.



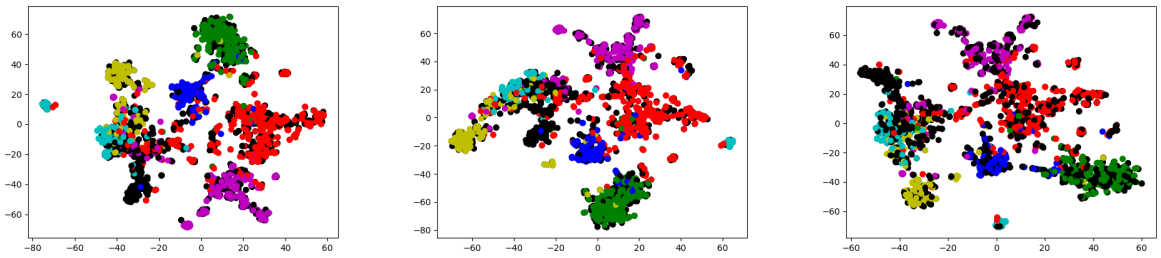
(a) 50% labeled alignment. (b) 30% labeled alignment. (c) 20% labeled alignment.

Figure 3.13: Precision@30 with different amount of labeled alignment.

Hyperparameters. For all the effectiveness evaluations, we set $\lambda = 0.1$, $K = 20$, $\alpha = 0.1$, $\alpha_1 = 0.01$, $\alpha_2 = 1$, $\rho = 1.2$. We use 0.001 as the learning rate. We keep the default values for all the hyperparameters used in the baseline methods.

Machines. The proposed method ORIGIN is implemented in Tensorflow [131] with Adam optimizer. We use one Nvidia Titan X with 12G RAM as GPU. The CPU-based methods are performed with four 3.6GHz Intel Cores and 32G RAM.

Effectiveness Results. We first evaluate the alignment accuracy of the proposed ORIGIN compared with the baseline methods. To compute the alignment accuracy, we conduct a greedy matching [3] as the post-processing step to obtain the one-to-one node mapping between two networks, followed by calculating the percentage of *all* ground-truths that can be correctly aligned as the alignment accuracy. The results are summarized in Figure 3.12. We have the following observations. First, our proposed method ORIGIN outperforms all the baseline methods. Specifically, our method can achieve an up to 5% improvement in terms of the alignment accuracy compared with FINAL-N, a strong baseline for attributed network alignment. Recall that FINAL-N can be viewed as a method based on the linear transformation to maximize a variant of Koopmans-Beckmann's QAP. This demonstrates the benefits of the non-rigid network alignment. Besides, our method can achieve better alignment results than other node representation-based methods, namely REGAL and IONE. Second,



(a) Cora-1 node embeddings. (b) Displaced cora-1. (c) Cora-2 node embeddings.

Figure 3.14: 2-D t-SNE visualization of node representations of networks cora-1 and cora-2.

our method achieves an at least 15% accuracy improvement compared with SageAlign, a variant of ORIGIN. This shows that the node representations of multiple networks jointly learned with the proposed Multi-GCN are more powerful for the specific alignment task. Besides, SageAlign itself also demonstrates the effectiveness of the proposed point set alignment algorithm to align node representations in the Euclidean space. Finally, as the amount of labeled alignment decreases, our method consistently outperforms other baseline methods.

To further verify if our method can correctly align unlabeled nodes, we compare the precision@30 score with the baseline methods. We define the precision@30 as follows. For any unlabeled node u in $\mathcal{G}_1$, if the correct alignment (say node v in $\mathcal{G}_2$) belongs to the top-30 most similar nodes to node u, we say there is a *hit*. We calculate the precision@30 by precision@30=(# of hits)/(# of unlabeled nodes). As Figure 3.13 shows, our method achieves higher precision@30 scores than baseline methods in most cases (i.e., our method is slightly lower than FINAL-N only in a few cases).

We also visualize in Figure 3.14 the node representations of networks cora-1 and cora-2 in 2-D space by t-SNE and demonstrate the benefits of point set alignment. Here, we use different colors as different node classes, which are known a priori as additional information, to help indicate the node correspondence. For example, nodes in purple in network cora-1 are expected to be aligned with the purple nodes in network cora-2. By comparing Figure 3.14 (a) (i.e., $\bar{\mathbf{X}}_1$) and Figure 3.14 (c) (i.e., $\bar{\mathbf{X}}_2$), we observe that despite minimizing the distances among node representations across networks by Eq. (3.56), nodes that ought to be aligned may still be far away from each other in the Euclidean space, such as the nodes in purple of two networks. This incomparability among node representations could further mislead the node alignment. In contrast, by our proposed non-rigid point set alignment, node representations of network cora-1 can be moved towards those of network cora-2. Consequently, the displaced node representations $\mathbf{X}_1^o$ shown in Figure 3.14 (b) are closer to the representations $\mathbf{X}_2^o$ of their aligned counterparts in network cora-2.

Moreover, we conduct a parameter study on cora-1 and cora-2 dataset about how the importance of different views used for point set alignment (i.e., α_1, α_2) influences the alignment accuracy. As Figure 3.15 shows, the alignment accuracy is stable over a wide range of α_1, α_2 ($0.01 \leq \alpha_1, \alpha_2 \leq 10$). Meanwhile, we observe that when $\alpha_2 > \alpha_1$, the proposed method achieves higher alignment accuracies than those when $\alpha_2 \leq \alpha_1$. This indicates the graph view indeed benefits point set alignment by calibrating the alignment based on a single view.

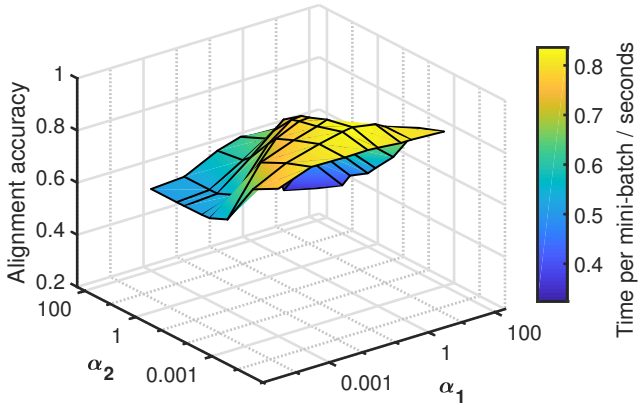


Figure 3.15: Parameter study on α_1, α_2.

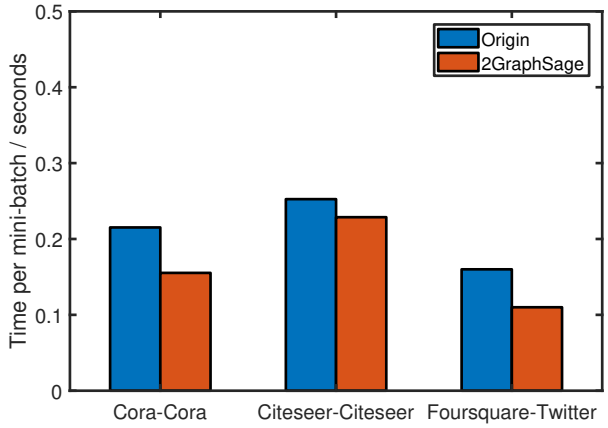


Figure 3.16: Running time per mini-batch.

Efficiency Results. We evaluate the efficiency of our proposed Multi-GCN for three alignment scenarios in terms of the running time per mini-batch. We fix each mini-batch with size 200 and compare our method (Multi-GCN) with its single-network counterpart (GraphSage). In particular, we measure the time cost of Multi-GCN minimizing Eq. (3.55) and the cost of GraphSage on two networks minimizing the loss Eq. (3.50). As shown in Figure 3.16, the time cost of the proposed Multi-GCN is very close to the GraphSage counterpart. This suggests that the extra computational cost for Inter-GCN, which brings the alignment improvement as shown above, is actually quite light.

3.3 NEURAL CROSS-NETWORK TRANSFORMATION

In the era of big data, networks are often multi-sourced. Finding the node associations across different networks is a key stepping stone to explore deep insights from such multi-sourced networks. If nodes in different networks represent the same type of entities, finding the cross-network node associations is essentially the *(soft) network alignment* problem [9]. For example, aligning suspects across different transaction networks helps integrate the transaction histories of the suspects at different financial institutes, which in turn facilitates

to uncover the complex financial fraud schema. On the other hand, if nodes in different networks represent different types of entities, finding the cross-network node associations is often referred to as the *cross-layer dependency inference* problem [132]. It indicates how entities of different types from different networks interact with each other. For instance, the user-product interactions across a social network and a product similarity network can be used for social recommendations [133]. In a biological system, the protein-protein interaction (PPI) networks are often coupled with the disease networks, and the cross-network node associations may indicate how diseases are related to different genes [134].

Traditional methods for cross-network node associations are often, explicitly or implicitly, built upon the linearity and/or consistency assumptions. For example, classic graph matching based network alignment methods often assume networks are noisy permutations of each other and minimize $\|\mathbf{A}_2 - \mathbf{S}'\mathbf{A}_1\mathbf{S}\|_F^2$ where $\mathbf{A}_1, \mathbf{A}_2$ are the adjacency matrices of two networks and $\mathbf{S}$ is the permutation matrix [6, 18]. This formulation together with many of its variants (e.g., [9]) implicitly embraces a *linear operation*[4]. As for cross-layer dependency inference, a typical approach is based on network-regularized matrix factorization [132, 135, 136] under the *consistency/homophily* assumption. For example, in social recommendation, these methods typically assume similar users tend to share similar latent representations.

More recent efforts aim to approach the cross-network node association problem by learning node embedding vectors of different networks [27, 137]. These methods can potentially go beyond the linearity and/or the consistency assumptions behind the complex cross-network node associations by learning node embedding vectors through nonlinear functions. However, the node embedding vectors of different networks often lie in the *disparate vector spaces* which might be incomparable with each other. For instance, if we shift, rotate or scale the node embeddings of one network, it could significantly impair alignment results [138].

In this work, we address the above limitations and tackle cross-network node associations from a new angle, i.e., cross-network transformation. We ask a generic question: *Given two different networks, how can we transform one network to another?* We design an end-to-end model NetTrans that bears three key advantages. First (*composite transformation*), instead of learning a single linear transformation function underpinning graph matching based methods, the proposed model learns a composition of nonlinear functions to transform both network structures and node attributes, and in the meanwhile unveils the cross-network associations. Second (*representation power*), by exploiting the multi-resolution characteristics

[4]To see this, the objective function of graph matching based network alignment is equivalent to minimizing $\|\operatorname{vec}(\mathbf{A}_2) - \tilde{\mathbf{S}}\operatorname{vec}(\mathbf{A}_1)\|_2^2$ where $\operatorname{vec}(\mathbf{A}_1), \operatorname{vec}(\mathbf{A}_2)$ denote the vectors of node pairs and $\tilde{\mathbf{S}} = \mathbf{S}' \otimes \mathbf{S}'$ is the Kronecker product of the permutation matrix. $\tilde{\mathbf{S}}$ is used as a *single* linear transformation across the node pairs of two networks.

underlying the networks, the proposed pooling layer TransPool can learn the hierarchical representations of the networks and the unpooling layer TransUnPool learns the transformations at different resolutions. Third (*generality*), the proposed model is generic and it can be easily applied to numerous tasks, such as network alignment, social recommendation, cross-layer dependence inference. The main contributions can be summarized as follows.

- *Problem Definition.* To our best knowledge, we are the first to address the cross-network transformation problem.
- *End-to-End Model.* We design an end-to-end model NetTrans which composes of the novel pooling and unpooling operations to learn the transformation functions and find the node associations across different networks.
- *Experimental Results.* We perform extensive experiments in network alignment and social recommendation, which demonstrate the effectiveness of the proposed model to find the cross-network node associations.

3.3.1 Problem Definition

Table 3.5 summarizes the main symbols and notations used throughout this study. We use bold uppercase letters for matrices (e.g., $\mathbf{A}$), bold lowercase letters for vectors (e.g., $\mathbf{a}$), and lowercase letters (e.g., α) for scalars. We use $\mathbf{A}(i,j)$ to denote the entry at the intersection of the i-th row and j-th column of the matrix $\mathbf{A}$, $\mathbf{A}(i,:)$ to denote the i-th row of $\mathbf{A}$ and $\mathbf{A}(:,j)$ to denote the j-th column of $\mathbf{A}$. We denote the transpose of a matrix by a superscript prime (e.g., $\mathbf{A}'$ is the transpose of $\mathbf{A}$). Furthermore, we use subscripts to index the matrices in different layers. For example, we use $\mathbf{A}_1^{(0)} = \mathbf{A}_1$ to denote the input adjacency matrix of $\mathcal{G}_1$ and $\mathbf{A}_1^{(l)}$ as the adjacency matrix of the coarsened network in the l-th layer. In addition, we use u, v to index the nodes of two input networks and use u', v' to index the supernodes of the coarsened networks in each layer. Note that the supernode-u' of the output coarsened network obtained in the $(l-1)$-th layer is equivalent to the node-u' of the input network in the l-th layer. In this work, we use 'graphs' and 'networks' interchangeably.

Multi-sourced networks are often associated with each other, in terms of the network structures, node attributes and cross-network node associations. In other words, there exist some functions that link the multi-sourced networks together. In this work, we consider to learn a transformation function denoted by $g(\cdot)$ such that the source network $\mathcal{G}_1$ can be transformed to the target network $\mathcal{G}_2$, i.e., $g(\mathcal{G}_1) \simeq \mathcal{G}_2$. Figure 3.17 (a) presents an illustrative example in the network alignment scenario. As we can see, the source network $\mathcal{G}_1$

Table 3.5: Symbols and notations.

Symbols	Definition
$\mathcal{G}_1$, $\mathcal{G}_2$	input source and target networks
$\mathcal{V}_1$, $\mathcal{V}_2$	the sets of nodes of $\mathcal{G}_1$ and $\mathcal{G}_2$
$\mathbf{A}_1, \mathbf{A}_2$	input adjacency matrices of $\mathcal{G}_1$ and $\mathcal{G}_2$
$\mathbf{X}_1, \mathbf{X}_2$	input node attribute matrices of $\mathcal{G}_1$ and $\mathcal{G}_2$
n_1, n_2	# of nodes in $\mathcal{G}_1$ and $\mathcal{G}_2$
$\mathbf{A}_1^{(l)}, \mathbf{A}_2^{(l)}$	adjacency matrices of $\mathcal{G}_1$ and $\mathcal{G}_2$ in the l-th layer
$\mathbf{X}_1^{(l)}, \mathbf{X}_2^{(l)}$	node feature matrices in the l-th layer
$\mathbf{P}_l$	node-supernode assignment matrix in the l-th layer
$n_1^{(l)}$, $n_2^{(l)}$	# of nodes in $\mathbf{A}_1^{(l)}$ and $\mathbf{A}_2^{(l)}$ in the l-th layer
L	# of layers of both encoder and decoder
α, β, γ	parameters controlling the importance of loss terms
$[\cdot\|\cdot]$	concatenation operator of two vectors

and target network $\mathcal{G}_2$ have different network structures as well as node attributes, and more importantly, the cross-network node associations (i.e., node correspondences) are unknown. Our goal is to learn the transformation functions on both network structures and node attributes while identifying the cross-network associations.

To be specific, we use the following major notations to describe the cross-network transformation. First, we denote the input source network $\mathcal{G}_1$ and target network $\mathcal{G}_2$ by triplets, i.e., $\mathcal{G}_1 = \{\mathcal{V}_1, \mathbf{A}_1, \mathbf{X}_1\}$ and $\mathcal{G}_2 = \{\mathcal{V}_2, \mathbf{A}_2, \mathbf{X}_2\}$[5] where $\mathcal{V}_1, \mathbf{A}_1, \mathbf{X}_1$ denote the set of nodes, adjacency matrix and node attributes of $\mathcal{G}_1$, respectively and similarly for $\mathcal{G}_2$. Second, we denote the transformation function on network structures and node attributes by g such that $(\mathbf{A}_2, \mathbf{X}_2) \simeq g(\mathbf{A}_1, \mathbf{X}_1)$. Lastly, the transformation function induces the node associations across different networks and is denoted by g_{node}. Figure 3.17 (b) shows an example of the corresponding transformation in terms of network structure and node attributes, as well as the node associations. Given the above notations, we formally define the cross-network transformation problem as follows.

Problem 3.4. Cross-Network Transformation.

Given: (1) input source network $\mathcal{G}_1 = \{\mathcal{V}_1, \mathbf{A}_1, \mathbf{X}_1\}$ and target network $\mathcal{G}_2 = \{\mathcal{V}_2, \mathbf{A}_2, \mathbf{X}_2\}$ where $\mathcal{V}_1, \mathcal{V}_2$ denote the nodes of $\mathcal{G}_1, \mathcal{G}_2$, $\mathbf{A}_1, \mathbf{A}_2$ are the adjacency matrices and $\mathbf{X}_1, \mathbf{X}_2$ are the node attribute matrices of $\mathcal{G}_1, \mathcal{G}_2$, and (2) prior knowledge of the cross-network node associations $\mathbf{L}$ where $\mathbf{L}(u, v)$ indicates whether node-u in $\mathcal{G}_1$ associates with node-v in $\mathcal{G}_2$.

Output: (1) the cross-network transformation function $g = \mathcal{F} \circ \mathcal{H}$ where $\mathcal{F}$ denotes a

[5]If the attributes are not available, one can simply set $\mathbf{X}_1, \mathbf{X}_2$ as identity matrices or manually extract structure-dependent attributes.

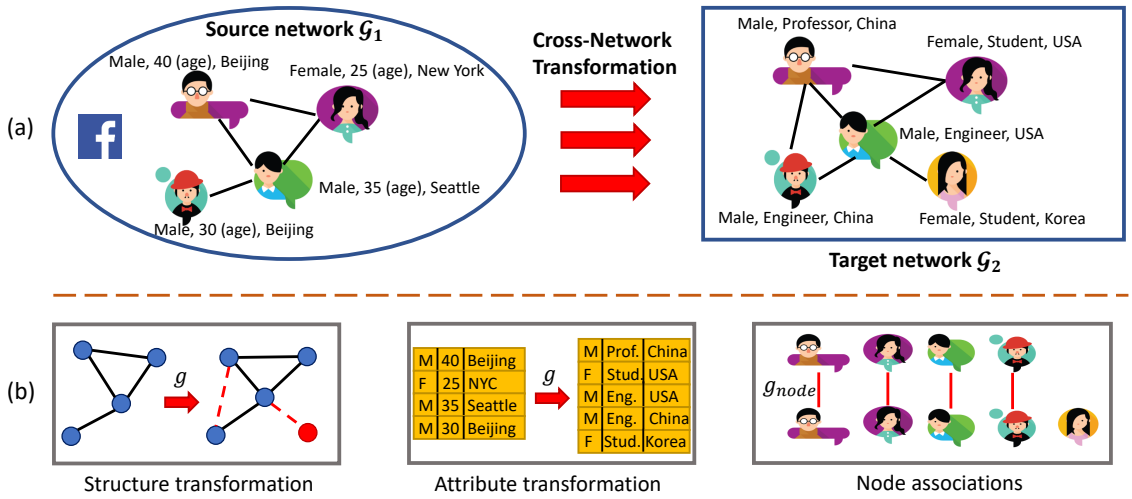


Figure 3.17: An illustration of cross-network transformation in the network alignment scenario. (a) shows the entire transformation across networks. (b) shows the transformations on network structure, node attributes and cross-network node associations.

set of encoding functions and $\mathcal{H}$ denotes the decoding functions such that $g(\mathcal{G}_1) \simeq \mathcal{G}_2$, and (2) the cross-network association function g_{node} where $g_{\text{node}}(u, v)$ measures to what extent node-u in $\mathcal{G}_1$ is associated with node-v in $\mathcal{G}_2$.

Let us take the graph matching based methods as an example to further illustrate the functions g and g_{node}, given two networks $\mathcal{G}_1 = \{\mathcal{V}_1, \mathbf{A}_1, \mathbf{X}_1\}$ and $\mathcal{G}_2 = \{\mathcal{V}_2, \mathbf{A}_2, \mathbf{X}_2\}$ with the same type of nodes, graph matching based methods aim to learn a permutation matrix $\mathbf{S} \in \mathbb{R}^{n_1 \times n_2}$, indicating the node correspondence between $\mathcal{V}_1, \mathcal{V}_2$ (i.e., g_{node}). In the meanwhile, the matrix $\mathbf{S}$ is also used as a single transformation function such that $\text{vec}(\mathbf{A}_2) \simeq \tilde{\mathbf{S}}\,\text{vec}(\mathbf{A}_1)$ and $\mathbf{X}_2 \simeq \mathbf{S}'\mathbf{X}_1$ where $\tilde{\mathbf{S}} = \mathbf{S}' \otimes \mathbf{S}'$. Thus, the transformation function g can be written as $g(\text{vec}(\mathbf{A}_1), \mathbf{X}_1) = (\tilde{\mathbf{S}}\,\text{vec}(\mathbf{A}_1), \mathbf{S}'\mathbf{X}_1)$. And the cross-network node association function $g_{\text{node}}(u, v) = \mathbf{S}(v, u)$. Despite its elegant mathematical formulation, such a single linear transformation might over-simplify the complex associations across networks.

Remarks. Note that the cross-network transformation problem is similar to but bears subtle differences from the graph-to-graph translation problem [139, 140]. For the latter, it implicitly assumes that nodes in different networks are of the same type and all node correspondences are perfectly *known a priori*. In contrast, the cross-network transformation problem that we study in this work aims to *learn* such node associations from the networks (i.e., the function g_{node}), in addition to learning the transformation function g.

We envision that the learned transformation functions from Problem 3.4 can be applied in a variety of data mining tasks. In this work, we focus on two such tasks, including (Task 1) network alignment and (Task 2) social recommendation. In Task 1, we consider two networks

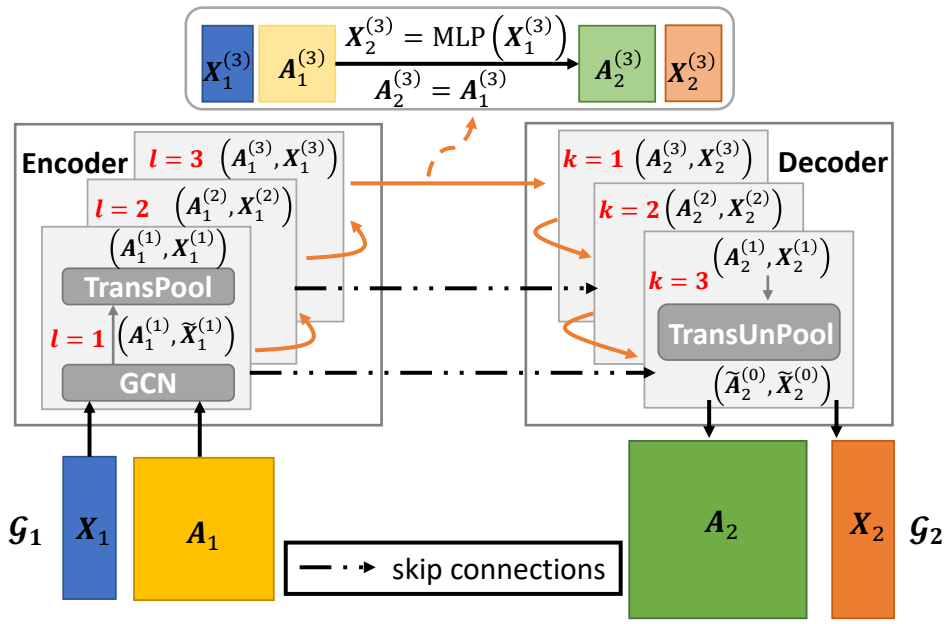


Figure 3.18: Overall architecture of the model NetTrans ($L = 3$).

with the same type of nodes and the transformation function g_{node} measures to what extent that two nodes are aligned with each other. In Task 2, we consider a social network and a product similarity network, and g_{node} predicts whether a user likes/buys a product.

3.3.2 The NetTrans Model

In this part, we present the proposed model NetTrans, an end-to-end semi-supervised model to solve Problem 3.4. We start by giving an overview of our model, and then detail the components of the model, followed by the discussions on the potential generalizations.

NetTrans Model Overview. The core challenge of cross-network transformation lies in how to design a model that can jointly learn the transformation function g and the cross-network node associations g_{node}. In this study, we design an encoder-decoder architecture that decomposes the transformation function g into two parts, including the encoder $\mathcal{F}$ and the decoder $\mathcal{H}$. We exploit the multi-resolution characteristics of real-world networks in both the encoder and the decoder. The overall architecture of the proposed NetTrans model is shown in Figure 3.18. The encoder $\mathcal{F}$ aims to coarsen the source network and learn the network structure and node representations at different resolutions. On the other hand, the decoder $\mathcal{H}$ reconstructs the structure and node representations of the target network at different resolutions. To make the source network and target network at different resolutions comparable to each other, we design the encoder and decoder to have the same number of

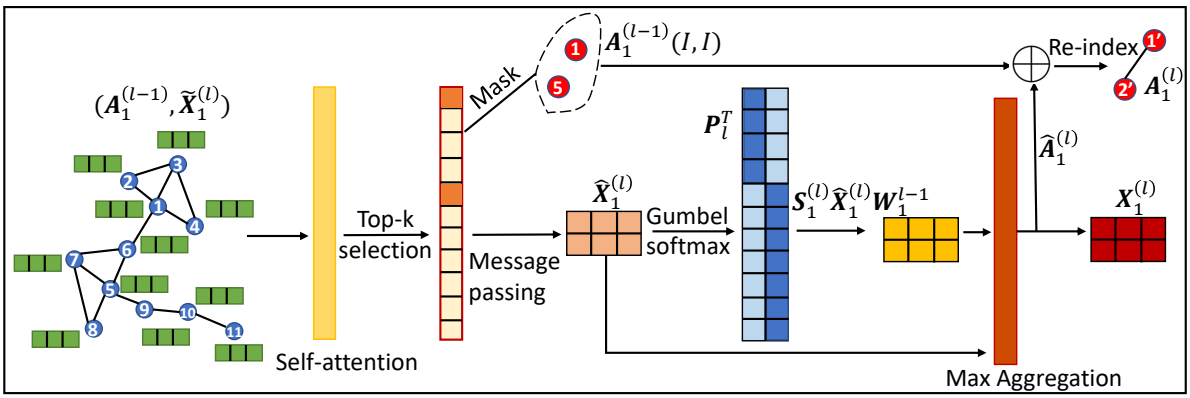


Figure 3.19: Description of the pooling layer (TransPool) in the l-th encoder layer.

layers (i.e., L). To learn g_{node}, we need to simultaneously learn the node assignments across two adjacent resolutions indicating which nodes at the finer resolution are merged into which node(s) in the next coarser resolution. To this end, we develop a pooling layer TransPool in the encoder and an unpooling layer TransUnPool in the decoder.

The intuition behind such a design is that we could simplify the cross-network transformation at the coarser resolutions, since the coarsened networks are likely to become more similar with each other. For instance, given a social network and a product similarity network, the nodes at the coarse resolutions might share similar latent meanings (e.g., a group of users who like to buy computers vs. a group of computer-related products). Moreover, the association between a group of users and a group of similar products will provide critical auxiliary information to infer the associations between the users and products in these groups. With this hierarchical learning, the proposed TransUnPool layers naturally learns to model how network structures and node representations are transformed at different resolutions.

NetTrans Encoder. Denote $\mathcal{F} = \{f_l\}$ as the functions in the encoder where f_l, $l = 1, \cdots, L$ represents the encoding function in the l-th encoder layer. In the l-th encoder layer, the function $f_l(\cdot, \cdot)$ on the network can be decomposed into learning the adjacency matrix and node attributes of the coarsened network. Denote $\mathbf{A}_1^{(l)} \in \mathbb{R}^{n_1^{(l)} \times n_1^{(l)}}$ and $\mathbf{X}_1^{(l)} \in \mathbb{R}^{n_1^{(l)} \times d_l}$ as the output adjacency matrix and node representations of the coarsened network in the l-th layer. Given the inputs $\mathbf{A}_1^{(l-1)} \in \mathbb{R}^{n_1^{(l-1)} \times n_1^{(l-1)}}$ $(n_1^{(l)} \leq n_1^{(l-1)})$ and $\mathbf{X}_1^{(l-1)} \in \mathbb{R}^{n_1^{(l-1)} \times d_{l-1}}$ which is the output coarsened network in the $(l-1)$-th layer, we can denote the encoding function by $(\mathbf{A}_1^{(l)}, \mathbf{X}_1^{(l)}) = f_l(\mathbf{A}_1^{(l-1)}, \mathbf{X}_1^{(l-1)})$. For example, the outputs of the first encoder layer can be computed by $(\mathbf{A}_1^{(1)}, \mathbf{X}_1^{(1)}) = f_1(\mathbf{A}_1^{(0)}, \mathbf{X}_1^{(0)})$.

To learn the structure of the coarsened networks, one prevalent choice is to coarsen the network with an assignment matrix, e.g., $\mathbf{A}_1^{(l)} = \mathbf{P}_l \mathbf{A}_1^{(l-1)} \mathbf{P}_l'$ where $\mathbf{P}_l \in \mathbb{R}^{n_1^{(l)} \times n_1^{(l-1)}}$ and

$\mathbf{P}_{l-1}(u', u)$ measures the strength of node-u in $\mathbf{A}_1^{(l-1)}$ being merged into the supernode-u'. Prior methods to compute $\mathbf{P}_l$ include the classic methods that calculate it deterministically (e.g., [141]) and graph neural networks based methods [142]. One advantage is that all nodes are assigned to certain supernodes based on the dense assignment matrix. However, these methods might lead to costly computations and the densely connected network structure [142]. We also remark in Proposition 3.1 that this coarsening process is built upon linear operations which might insufficiently capture the underlying hierarchical structures.

Proposition 3.1. Given an assignment matrix $\mathbf{P}_l$, the coarsening process $\mathbf{A}_1^{(l)} = \mathbf{P}_l \mathbf{A}_1^{(l-1)} \mathbf{P}_l'$ constructs edges by linear operations.

Proof. By eigenvalue decomposition on $\mathbf{A}_1^{(l-1)}$, $\mathbf{A}_1^{(l-1)} = \mathbf{U}_1 \mathbf{\Lambda}_1 \mathbf{U}_1'$, we have

$$\mathbf{A}_1^{(l)}(u_1', u_2') = (\mathbf{P}_l \mathbf{U}_1 \mathbf{\Lambda}_1 \mathbf{U}_1' \mathbf{P}_l')(u_1, u_2) = [\mathbf{P}_l(u_1', :)\mathbf{U}_1]\mathbf{\Lambda}_1[\mathbf{P}_l(u_2', :)\mathbf{U}_1]' \tag{3.69}$$

where $\mathbf{U}_1 \in \mathbb{R}^{n_1^{(l-1)} \times r}$ and $r < n_{l-1}$ only if $\mathbf{A}_1^{(l-1)}$ is low-rank. In this way, by considering $\mathbf{U}_1$ as node representations, the existence of an edge and its weight between supernode u_1' and supernode u_2' is determined equivalently by first computing supernodes' representations by linear aggregations based upon assignment matrix $\mathbf{P}_l$, and then a weighted inner product between the representations of supernodes u_1', u_2'. Both steps only involve linear operations on node representations. QED.

Another way to coarsen networks is the learnable importance-based pooling operations [143, 144]. These methods basically select the top-k important nodes as *supernodes*, and preserve the original connections among the selected nodes as the edges among the corresponding supernodes. The advantages of these methods include the efficient computations and the sparse structure of the coarsened networks. Note that keeping the original connections can be viewed as a special case of assignment matrix based methods. Specifically, the assignment matrix $\mathbf{P}_l(u', u) = 1$ if and only if node-u is selected and re-indexed to supernode-u'. Thus, according to Proposition 3.1, the informativeness of the coarsened structure by these methods is also hindered by the underlying linear operations. In addition, it is unknown how those unselected nodes are assigned to the supernodes (i.e., $\mathbf{P}_l(:, u) = 0$ for all u that are not selected). To this end, we design TransPool (shown in Figure 3.19) that can balance between the above two strategies and learn the assignments for all nodes.

Following [143, 144], in the l-th encoder layer, we first feed the inputs $(\mathbf{A}_1^{(l-1)}, \mathbf{X}_1^{(l-1)})$ to a graph convolutional layer (e.g., [122]) before the pooling layer. By denoting $\tilde{\mathbf{X}}_1^{(l)} = \text{GCN}(\mathbf{A}_1^{(l-1)}, \mathbf{X}_1^{(l-1)})$, we compute the self-attention scores $\mathbf{z}_l \in \mathbb{R}^{n_1^{(l-1)}}$ by a graph convolu-

tional layer [122] to measure the node importance [144]. This is formulated by

$$\mathbf{z}_l = \sigma\left(\tilde{\mathbf{D}}_{l-1}^{-\frac{1}{2}}\tilde{\mathbf{A}}_1^{(l-1)}\tilde{\mathbf{D}}_{l-1}^{-\frac{1}{2}}\tilde{\mathbf{X}}_1^{(l)}\mathbf{W}_l^{\text{self}}\right) \tag{3.70}$$

where $\sigma(\cdot)$ denotes the nonlinear activation function (e.g., tanh), $\tilde{\mathbf{A}}_1^{(l-1)} = \mathbf{A}_1^{(l-1)} + \mathbf{I}$, $\mathbf{A}_1^{(l-1)} \in \mathbb{R}^{n_1^{(l-1)} \times n_1^{(l-1)}}$ is the input adjacency matrix in the l-th encoder layer, $\tilde{\mathbf{D}}_{l-1} = \text{diag}(\tilde{\mathbf{A}}_1^{(l-1)}\mathbf{1})$ is the diagonal degree matrix of $\tilde{\mathbf{A}}_1^{(l-1)}$, $\tilde{\mathbf{X}}_1^{(l)} \in \mathbb{R}^{n_1^{(l-1)} \times d_l}$ is the input node feature matrix, and $\mathbf{W}_l^{\text{self}} \in \mathbb{R}^{d_l}$ contains the parameters to compute the self-attention scores. By using these self-attention scores to measure node importance, both network structure and node features are naturally encoded. Thus, nodes of the top-$n_1^{(l)}$ scores are likely to be more important to capture the structural and feature information. In [143, 144], these selected top-$n_1^{(l)}$ nodes are then used as masks to construct the adjacency matrix and node features of the coarsened network. Specifically, by denoting the indices of the selected nodes as $\mathcal{I} = \text{top-rank}(\mathbf{z}_l, n_1^{(l)})$, [143, 144] compute $\mathbf{A}_1^{(l)} = \mathbf{A}_1^{(l-1)}(\mathcal{I}, \mathcal{I}) = \mathbf{P}_l\mathbf{A}_1^{(l-1)}\mathbf{P}_l'$ where $\mathcal{I}_{u'}$ is the u'-th element of $\mathcal{I}$ and $\mathbf{P}_l(u', \mathcal{I}_{u'}) = 1,\ \forall u' = 1, \cdots, n_1^{(l)}$ are the only nonzero elements in $\mathbf{P}_l$. The coarsened node features are computed by $\mathbf{X}_1^{(l)}(u', :) = \tilde{\mathbf{X}}_1^{(l)}(\mathcal{I}_{u'}, :) \odot \mathbf{z}_l(u')$ where $\odot$ is element-wise product.

However, this simple masking-based pooling operation has two potential limitations. First, the computation of output feature matrix $\mathbf{X}_1^{(l)}$ in [143, 144] insufficiently leverages the representations of the unselected nodes by simply re-scaling $\tilde{\mathbf{X}}_1^{(l)}$ based on $\mathbf{z}_l$. Second, as mentioned before, edges in the coarsened adjacency matrix are constructed by linear operations. Besides, the coarsened network empirically may contain isolated nodes. For example, an isolated node-u' can occur when $\mathbf{A}_1^{(l-1)}(\mathcal{I}_{u'}, \mathcal{I} \setminus \mathcal{I}_{u'}) = 0$ for some u' (e.g., node-1 and node-5 in red in Figure 3.19). Note that the isolated nodes cannot be completely avoided by Eq. (3.70). Such an issue could further lead to the inability of the information propagation to the isolated nodes in the next encoder layer, making the network structure and node representation learning at the coarser resolutions even worse.

To address the first issue, instead of directly rescaling the representation vectors, we allow message passing from $n_1^{(l-1)}$ nodes to the selected $n_1^{(l)}$ supernodes. In particular, we use attention-based message passing [145] formulated as below.

$$\hat{\mathbf{X}}_1^{(l)}(u', :) = \sigma\left(\tilde{\mathbf{X}}_1^{(l)}(\mathcal{I}_{u'}, :)\mathbf{W}_l^1 + \sum_{u \in \mathcal{N}_{u'}} \alpha_{u'u}\tilde{\mathbf{X}}_1^{(l)}(u, :)\mathbf{W}_l^1\right) \tag{3.71}$$

$$\alpha_{u'u} = \frac{\exp\left(\mathbf{a}_l'\left[\tilde{\mathbf{X}}_1^{(l)}(\mathcal{I}_{u'}, :)\mathbf{W}_l^1 \| \tilde{\mathbf{X}}_1^{(l)}(u, :)\mathbf{W}_l^1\right]\right)}{\sum_{u_1 \in \mathcal{N}_{u'}} \exp\left(\mathbf{a}_l'\left[\tilde{\mathbf{X}}_1^{(l)}(\mathcal{I}_{u'}, :)\mathbf{W}_l^1 \| \tilde{\mathbf{X}}_1^{(l)}(u_1, :)\mathbf{W}_l^1\right]\right)} \tag{3.72}$$

where $\mathcal{N}_{u'}$ denotes the 1-hop neighborhood of supernode-u' in the bipartite graph $\mathcal{G}_b$ formed by $\mathbf{A}_1^{(l-1)}(:, \mathcal{I})$, $\mathbf{a}_l \in \mathbb{R}^{2d_l}$ and $\mathbf{W}_l^1 \in \mathbb{R}^{d_l \times d_l}$ are the parameters to be learned. However, Eq. (3.71) cannot aggregate the features from the nodes that are multi-hop away from the selected nodes (e.g., from node-10 to node-5). As a remedy, we additionally aggregate the node features based on the assignment matrix $\mathbf{P}_l$. To efficiently learn $\mathbf{P}_l$, we propose the following mechanism to select supernode candidates. First, for a node-u that have some supernodes as their 1-hop neighbors in $\mathcal{G}_b$, they can be assigned only to their 1-hop neighboring supernodes, i.e., $\mathcal{C}(u) = \{u' | \mathbf{A}_1^{(l-1)}(u, \mathcal{I}_{u'}) = 1, u' = 1, \cdots, n_1^{(l)}\}$. For example, in Figure 3.19, node-6 can be assigned to either node-1 or node-5. Second, for some node-u that connects to supernodes exactly in 2 hops (e.g., node-10 in Figure 3.19), we select the candidates of supernodes by $\mathcal{C}(u) = \{u' | \mathbf{A}_1^{(l-1)}(u, :)\mathbf{A}_1^{(l-1)}(:, \mathcal{I}_{u'}) = 1, \mathbf{A}_1^{(l-1)}(u, \mathcal{I}_{u'}) = 0, u' = 1, \cdots, n_1^{(l)}\}$. For the rest of nodes (e.g., node-11), we simply set $\mathcal{C}(u) = \{u' | u' = 1, \cdots, n_1^{(l)}\}$. In addition, a hard assignment matrix $\mathbf{P}_l$ often requires each column of $\mathbf{P}_l$ to be a one-hot vector, i.e., $\mathbf{P}_l(u', u) = 1$ if and only if node u is merged into supernode u'. However, it is very difficult to directly learn those one-hot vectors as they essentially involve discrete variables, making the computations non-differentiable. In this work, we use the continuous Gumbel softmax [146] functions to approximate them which computes $\mathbf{P}_l$ by

$$\mathbf{P}_l(u', u) = \frac{\exp\left(\left[\log\left(\hat{\mathbf{X}}_1^{(l)}(u', :)\mathbf{W}_l^g(\tilde{\mathbf{X}}_1^{(l)})'\right) + g_{u'u}\right]/\tau\right)}{\sum_{c \in \mathcal{C}(u)} \exp\left(\left[\log\left(\hat{\mathbf{X}}_1^{(l)}(c, :)\mathbf{W}_l^g(\tilde{\mathbf{X}}_1^{(l)})'\right) + g_{cu}\right]/\tau\right)} \tag{3.73}$$

where $u' \in \mathcal{C}(u)$, $g_{u'u}$ is drawn from Gumbel$(0, 1)$ distribution, $\mathbf{W}_l^g \in \mathbb{R}^{d_l \times d_l}$ is the parameter matrix and τ is the softmax temperature. According to the Gumbel softmax distribution [146], as $\tau \to +\infty$, $\mathbf{P}_l(:, u)$ becomes a uniform distribution. In contrast, as $\tau \to 0$, $\mathbf{P}_l(:, u)$ is close to a one-hot vector but the variance of the gradients is large. Thus, we learn the parameters by starting with a large temperature and annealing to a small temperature τ. We then aggregate the information from distant nodes into the supernodes by $\mathbf{P}_l\tilde{\mathbf{X}}_1^{(l)}\mathbf{W}_l^1$, which can aid the representation learning of supernodes to better summarize the local neighborhoods. The output representations of the supernodes are computed by

$$\mathbf{X}_1^{(l)} = \text{Aggregate}(\hat{\mathbf{X}}_1^{(l)}, \mathbf{P}_l\tilde{\mathbf{X}}_1^{(l)}\mathbf{W}_l^1) \tag{3.74}$$

where Aggregate$(\cdot, \cdot)$ is a layer-wise aggregation. In this work, we use the max aggregation which simply takes the element-wise max.

To go beyond the linearity behind the coarsening process and address the issue of isolated

supernodes, our key idea is to leverage the representations of supernodes to add auxiliary weighted edges. Specifically, we add weights to the existing edges among the supernodes in $\mathbf{A}_1^{(l-1)}(\mathcal{I},\mathcal{I})$ to measure the edge strengths. For isolated supernodes, we compensate the edges by adding weighted edges only between isolated supernodes and the rest of supernodes. Note that the isolated supernodes $\mathcal{I}_l^s$ can be simply detected by whether there exist edges connecting to them in $\mathbf{A}_1^{(l-1)}(\mathcal{I},\mathcal{I})$ [147]. Then, under the classic assumption that nodes with similar representations are likely to be connected, the auxiliary edges to be added are computed by a sigmoid function, i.e., $\mathbf{A}_1^{(l)} = \frac{1}{2}(\mathbf{A}_1^{(l)}(\mathcal{I},\mathcal{I}) + \hat{\mathbf{A}}_1^{(l)})$ where

$$\hat{\mathbf{A}}_1^{(l)}(u_1', u_2') = \begin{cases} 2\sigma_s\left(\mathbf{X}_1^{(l)}(u_1',:)\mathbf{X}_1^{(l)}(u_2',:)'\right) & \text{if } u_1' \in \mathcal{I}_l^s \text{ or } u_2' \in \mathcal{I}_l^s \\ \sigma_s\left(\mathbf{X}_1^{(l)}(u_1',:)\mathbf{X}_1^{(l)}(u_2',:)'\right) & \text{if } u_1' \notin \mathcal{I}_l^s \text{ and } u_2' \notin \mathcal{I}_l^s \\ 0 & \text{otherwise} \end{cases} \tag{3.75}$$

and $\sigma_s(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function. In summary, we have the encoding function on adjacency matrices and node features as

$$(\mathbf{A}_1^{(l)}, \mathbf{X}_1^{(l)}) = f_l(\mathbf{A}_1^{(l-1)}, \mathbf{X}_1^{(l-1)}) = \text{TransPool}\left(\text{GCN}(\mathbf{A}_1^{(l-1)}, \mathbf{X}_1^{(l-1)})\right) \tag{3.76}$$

NetTrans Decoder. Our goal of the decoder is to learn node representations and the edges among the nodes in the context of the target network $\mathcal{G}_2$. We denote the decoder by a set of functions $\mathcal{H} = \{h_k\},\ k = 1, \cdots, L$ where $h_k(\cdot,\cdot)$ represents the decoding function at the k-th decoder layer. At the k-th decoder layer, the decoding function takes $\mathbf{A}_2^{(L-k+1)}, \mathbf{X}_2^{(L-k+1)}$ as inputs and outputs $\mathbf{A}_2^{(L-k)}, \mathbf{X}_2^{(L-k)}$ as the adjacency matrix and node representations of the target network at the next finer resolution. Note that we have $\mathbf{X}_2^{(L-k+1)} \in \mathbb{R}^{n_2^{(L-k+1)} \times d_{L-k+1}}$ and $\mathbf{X}_2^{(L-k)} \in \mathbb{R}^{n_2^{(L-k)} \times d_{L-k}}$ where $n_2^{(L-k)}, n_2^{(L-k+1)}$ $(n_2^{(L-k)} \geq n_2^{(L-k+1)})$ denote the numbers of nodes. Similar to the encoder, we denote the k-th decoder layer as $(\mathbf{A}_2^{(L-k)}, \mathbf{X}_2^{(L-k)}) = h_k(\mathbf{A}_2^{(L-k+1)}, \mathbf{X}_2^{(L-k+1)})$.

Prior unpooling operator includes gUnPool [143] for a single network that restores the structure and node hidden representations of the input network $\mathcal{G}_1$ obtained in different encoder layers. However, it is restricted to a single network and cannot be applied to the cross-network scenario due to the following reasons. First (*node associations*), nodes in different networks can have different types. And even for the networks of the same node type, the cross-network node correspondences are unknown. Thus, without the knowledge of the cross-network node associations, it is inappropriate to use the node ordering of the source network as the reference of the target network. Second (*network structure*), networks from different sources might have different structural patterns. In this way, it might mislead

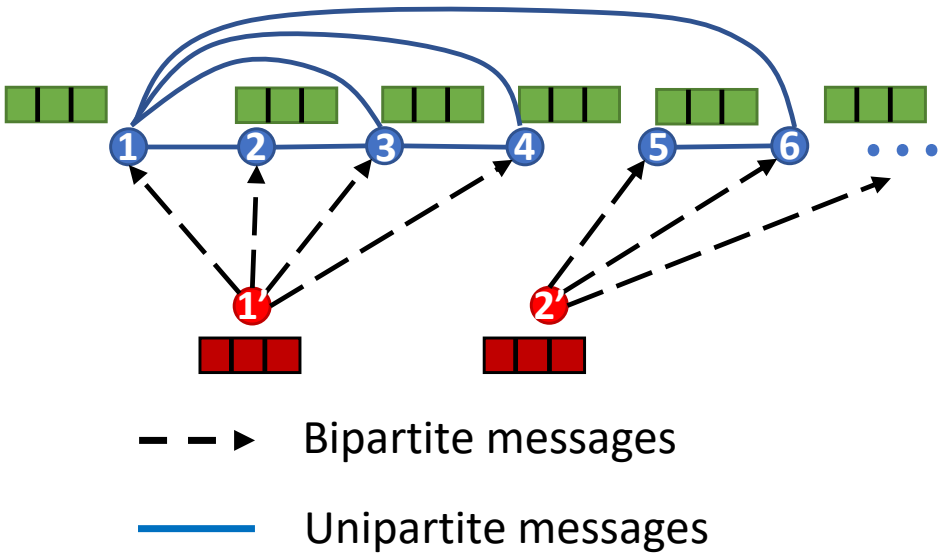


Figure 3.20: Illustrations of the message passing in TransUnPool layer corresponding to the pooling layer in Figure 3.19.

learning the structures of the target network at different resolutions. Third (*node representations*), nodes in different networks, either of the same type or of different types, can carry different structural and attribute information.

In this work, instead of *copying* the structure and node representations of source network, we design a novel unpooling layer (TransUnPool) to decode the target network at different resolutions. To address the first issue, since learning the cross-network node associations in different layers is nested together leading to a sophisticated learning process and costly computations, we simplify it by assuming the supernodes in the encoder layers represent the same set of latent entities as those in the corresponding decoder layers. For example, suppose a supernode-u' of the source social network in an encoder layer represents a group of users who like computers, then this supernode in its symmetric decoder layer may represent a set of products related to computers. That is, the supernode-u' in both encoder and decoder layers represents the same latent entity 'computer'. By doing this, the supernodes in the encoders and decoders are naturally one-to-one mapped and the assignment matrix $\mathbf{P}_l$ ($l \geq 2$) can be shared with the k-th ($k = L - l + 1$) decoder layer.

To address other issues, we hypothesize that two networks are close to each other at the coarsest resolution such that they share the same network structure and the node representations $\mathbf{X}_2^{(L)}$ can be transformed from $\mathbf{X}_1^{(L)}$ via a multilayer perceptron (MLP), i.e.,

$$\mathbf{A}_2^{(L)} = \mathbf{A}_1^{(L)}, \quad \mathbf{X}_2^{(L)} = \text{MLP}_1(\mathbf{X}_1^{(L)}). \tag{3.77}$$

Then, given the input supernode representations $\mathbf{X}_2^{(L-k+1)}$ $(k < L)$ in the k-th decoder layer, to learn the structure and node representations of the target network at the corresponding resolution, we design the following message passing module (shown in Figure 3.20) as a building block. Specifically, we define two types of messages that propagate to the nodes.

The first type of messages are those that propagate from supernodes to nodes via the bipartite edges $\mathbf{P}_{L-k+1}$ (denoted by black dashed lines in Figure 3.20). Mathematically, these bipartite messages are formulated by

$$\mathbf{m}_{v' \to v}^{k} = \mathbf{P}_{L-k+1}(v', v) \odot (\mathbf{X}_2^{(L-k+1)}(v', :)\mathbf{W}_k^2) \tag{3.78}$$

where $\mathbf{W}_k^2 \in \mathbb{R}^{d_{L-k+1} \times d_{L-k}}$ is the parameter matrix and $\mathbf{P}_{L-k+1}(v', v)$ is used to weigh the importance of the message based on to what extent that the node-v is merged into supernode-v' in the $(L-k+1)$-th encoder layer. Another type of messages are passed among the nodes in the unipartite graph. Our intuition is that the structures and node representations of the coarsened source networks in the encoder layers can provide some initial information, based on which we aim to calibrate the structure and node representations to fit the target network $\mathcal{G}_2$. Specifically, we first transfer the adjacency matrix $\mathbf{A}_1^{(L-k)}$ and node representations $\mathbf{X}_1^{(L-k)}$ through the skip connections (denoted by the black dash dotted lines in Figure 3.18). Then, we define the messages along the edges in the unipartite graph by

$$\mathbf{m}_{v_1 \to v}^{k} = \frac{1}{\sqrt{|\mathcal{N}_v|}\sqrt{|\mathcal{N}_{v_1}|}} \mathbf{X}_1^{(L-k)}(v_1, :)\mathbf{W}_k^3 \tag{3.79}$$

where $\mathbf{W}_k^3 \in \mathbb{R}^{d_{L-k+1} \times d_{L-k}}$ is the parameter matrix and $|\mathcal{N}_v|$ denotes the number of neighbors of node-v according to $\mathbf{A}_1^{(L-k)}$. In this way, the representations of nodes in the k-th decoder layer can be computed by combining both types of messages as

$$\mathbf{X}_2^{(L-k)}(v, :) = \sum_{\substack{v',\ s.t. \\ \mathbf{P}_{L-k+1}(v', v) > 0}} \mathbf{m}_{v' \to v}^{k} + \sum_{v_1 \in \mathcal{N}_v} \mathbf{A}_1^{(L-k)}(v_1, v) \odot \mathbf{m}_{v_1 \to v}^{k} \tag{3.80}$$

where $\mathbf{A}_1^{(L-k)}(v_1, v)$ denotes the weight of the edge (v_1, v).

To calibrate network structure to learn the structural pattern of $\mathcal{G}_2$ at different resolutions, we use $\mathbf{X}_2^{(L-k)}$ to compute to what extent we need to add/delete edges upon $\mathbf{A}_1^{(L-k)}$ by

$$\mathbf{A}_2^{(L-k)}(v, v_1) = \frac{1}{2} \max\{0, \mathbf{A}_1^{(L-k)}(v, v_1) + \sigma_t(\mathbf{X}_2^{(L-k)}(v, :)(\mathbf{X}_2^{(L-k)}(v_1, :))')\} \tag{3.81}$$

where $\sigma_t(x) \in (-1, 1)$ denotes the tanh activation function and we use a ReLU function to

make $\mathbf{A}_2^{(L-k)}$ only contain non-negative entries. However, Eq. (3.81) calculates $O(n_2^{(L-k)} \times n_2^{(L-k)})$ number of values, which is computationally costly. To make it more efficient, we only compute $\sigma_t(\mathbf{X}_2^{(L-k)}(v,:)(\mathbf{X}_2^{(L-k)}(v_1,:))')$ for the (v, v_1) such that $\mathbf{A}_2^{(L-k)}(v, v_1) \neq 0$, which in practice performs well in the experiments.

In the last decoder layer (i.e., $k = L$), we cannot directly use $\mathbf{P}_1$ as in Eq. (3.78) given the fact that nodes in $\mathcal{G}_1$ might either (1) have a different type from nodes in $\mathcal{G}_2$ or (2) have the same type but the correspondences to nodes in $\mathcal{G}_2$ are unknown. Fortunately, we have a partial knowledge of the cross-network node associations across $\mathcal{G}_1$ and $\mathcal{G}_2$ based on $\mathbf{L}(u, v)$ indicating whether node-u in $\mathcal{G}_1$ associates with node-v in $\mathcal{G}_2$ a priori. Note that $(\mathbf{P}_1\mathbf{L})(v', v) = \sum_{i=1}^{n_1} \mathbf{P}_1(v', u_i)\mathbf{L}(u_i, v)$ measures the strength of the assignment between node-v in $\mathcal{G}_2$ and the supernode-v' based on how many nodes in $\mathcal{G}_1$ that are associated with node-v in $\mathcal{G}_2$ a priori and also assigned to supernode-v'. In this way, we can use $\mathbf{Q}_1 = \mathbf{P}_1\mathbf{L}$ as the partially existing edges for bipartite message passing (i.e., dashed lines in Figure 3.20). In addition, we can construct bipartite messages at the last decoder layer from the nodes in $\mathcal{G}_1$ to nodes in $\mathcal{G}_2$ at the finest resolution through the prior knowledge $\mathbf{L}$ similarly as

$$\begin{aligned}
\mathbf{m}_{v'\to v}^L &= \mathbf{Q}_1(v', v) \odot (\mathbf{X}_2^{(1)}(v', :)\mathbf{W}_L^2) \\
\mathbf{m}_{v_1\to v}^L &= \frac{1}{\sqrt{|\mathcal{N}_v|}\sqrt{|\mathcal{N}_{v_1}|}}\mathbf{X}_2(v_1, :)\mathbf{W}_L^3 \\
\mathbf{m}_{u\to v}^L &= \mathbf{L}(u, v) \odot (\tilde{\mathbf{X}}_1^{(1)}(u, :)\mathbf{W}_L^4)
\end{aligned} \tag{3.82}$$

where $\mathcal{N}_v$ denotes the neighborhood of node-v and the node-v itself in the original target network $\mathcal{G}_2$. The final node representations of $\mathcal{G}_2$ can be computed by aggregating the messages in Eq. (3.82) as

$$\tilde{\mathbf{X}}_2^{(0)}(v, :) = \sum_{\substack{v',\ s.t. \\ \mathbf{Q}_1(v',v)>0}} \mathbf{m}_{v'\to v}^L + \sum_{v_1\in\mathcal{N}_v} \mathbf{A}_2(v_1, v) \odot \mathbf{m}_{v_1\to v}^L + \sum_{\substack{u,\ s.t. \\ \mathbf{L}(u,v)>0}} \mathbf{m}_{u\to v}^L \tag{3.83}$$

In summary, the k-th decoder layer can be computed by Eq. (3.80) and Eq. (3.81) (as well as Eq. (3.83) in the L-th decoder layer) and

$$h_k = \text{TransUnPool}\left(\mathbf{X}_2^{(L-k+1)}, \mathbf{A}_1^{(L-k)}, \mathbf{X}_1^{(L-k)}, \mathbf{P}_{L-k+1}\right) \tag{3.84}$$

NetTrans Model Training. With L encoder layers and L decoder layers, we can write the transformation function g of network structure and node attributes as

$$g = h_L \circ \cdots \circ h_1 \circ f_L \circ \cdots \circ f_1. \tag{3.85}$$

To learn the model parameters, our objectives are to reconstruct the target network $\mathcal{G}_2$ in terms of both structure and node attributes, while reflecting the observed cross-network node associations $\mathbf{L}$. To reconstruct the structure of $\mathcal{G}_2$, we minimize the binary cross-entropy loss over edges written as follows.

$$\mathcal{L}_{\text{adj}} = -\frac{1}{|\mathcal{E}|} \sum_{(v,v_1)\in\mathcal{E}} [y_{v,v_1} \log p_{v,v_1} + (1 - y_{v,v_1}) \log (1 - p_{v,v_1})] \tag{3.86}$$

where $p_{v,v_1} = \sigma_s(\tilde{X}_2^{(0)}(v,:)\tilde{X}_2^{(0)}(v_1,:)')$, $\mathcal{E} = \mathcal{E}_2 \cup \bar{\mathcal{E}}_2$ denotes the set of existing edges and samples of non-existent edges of $\mathcal{G}_2$ respectively, and $y_{v,v_1} = 1$ if $(v, v_1) \in \mathcal{E}_2$ otherwise 0.

To reconstruct node attributes $\mathbf{X}_2$ of $\mathcal{G}_2$, we further feed the output node representations $\tilde{\mathbf{X}}_2^{(0)}$ to an MLP and minimize the mean squared error with the input node attributes $\mathbf{X}_2$.

$$\mathcal{L}_{\text{attr}} = \frac{1}{n_2} \|\mathbf{X}_2 - \text{MLP}_2(\tilde{\mathbf{X}}_2^{(0)})\|_F^2 \tag{3.87}$$

In addition, we minimize the error of the known cross-network node associations by a margin ranking loss in the network alignment task and by a Bayesian personalized ranking loss [148] in social recommendation. Specifically in network alignment, given a set of triplets $\mathcal{O} = \{(u, v, v_1)|(u, v) \in \mathcal{R}^+, (u, v_1) \in \mathcal{R}^-\}$ where $\mathcal{R}^+ = \{(u, v)|\mathbf{L}(u, v) = 1\}$ denotes the observed node associations, and $\mathcal{R}^- = \{(u, v_1)|\mathbf{L}(u, v_1) = 0, \exists v,\ s.t.\ \mathbf{L}(u, v) = 1\}$ denotes a set of sampled negative associations, the margin ranking loss is

$$\mathcal{L}_{\text{rank}} = \frac{1}{|\mathcal{O}|} \sum_{(u,v,v_1)\in\mathcal{O}} \max\{0, \lambda - (g_{\text{node}}(u, v) - g_{\text{node}}(u, v_1))\} \tag{3.88}$$

where λ is the margin size and $g_{\text{node}}(u, v)$ is computed by

$$g_{\text{node}}(u, v) = \left[\mathbf{P}_1' \mathbf{X}_2^{(1)} (\tilde{\mathbf{X}}_2^{(0)})')\right](u, v) = \sum_{u'} \mathbf{P}_1(u', u) \left(\mathbf{X}_2^{(1)} (\tilde{\mathbf{X}}_2^{(0)})'\right)(u', v). \tag{3.89}$$

The overall loss function can be now formulated as below.

$$\mathcal{L} = \alpha\mathcal{L}_{\text{adj}} + \beta\mathcal{L}_{\text{attr}} + \gamma\mathcal{L}_{\text{rank}} \tag{3.90}$$

NetTrans: Variants and Generalizations. The proposed NetTrans is flexible and can be generalized in multiple aspects. Here, we give a few examples.

- **Bi-directional cross-network transformation.** NetTrans can be generalized to a bi-directional transformation model. That is, in addition to transforming from

the source network to the target network, the bi-directional model also learns the transformation functions in the reverse direction.

- **Graph-to-subgraph transformation.** When input source network is a large data graph and the target network is a small query graph, NetTrans can be tailored to learn the transformation from the data graph to the query graph and the node associations may indicate the subgraph matching.

- **Dynamic network transformation.** When we have a dynamic network $\mathcal{G}$, we can consider $\mathcal{G}'$ at timestamp t as the source network and $\mathcal{G}^{t+1}$ as the target network. In this case, we can generalize our transformation model to handle dynamic networks and learn how networks evolve over time.

- **Single network auto-encoder.** When the source network and target network are the same network in which case the node associations are naturally known, the proposed NetTrans model degenerates to an auto-encoder which captures the hierarchical structure of the network.

3.3.3 Experimental Evaluations

We apply the proposed model to network alignment and one-class social recommendation. We evaluate it in the following aspects:

- How accurate is our proposed transformation model for network alignment and recommendation?

- How does our model benefit from the proposed TransPool and TransUnPool layers?

Experimental Setup. We introduce the experimental setups as follows[6].
Datasets. The statistics of datasets are summarized in Table 3.6. The details of the datasets are as follows:

- *Cora citation network*: This dataset contains a citation network where nodes represent documents and edges represent the citations among documents. Each document has a binary feature vector represented by bag-of-words [149].

- *ACM co-author network*: This dataset was collected in 2016 including 2,381,688 papers with the author and venue information of each paper [150]. A co-author network was

[6]The code can be found at `https://github.com/sizhang92/NetTrans-KDD20`.

then extracted based on the papers published in four areas (DM, ML, DB and IR) in [12]. Nodes in the co-author network represent authors and edges indicate the co-authorship. The numbers of papers published by an author in 17 venues are used as the node attributes.

- *DBLP co-author network*: This dataset was collected in 2016 and it contains 3,272,991 papers. A co-author network was extracted in [12] similarly to the ACM dataset.

- *Foursquare*: This dataset contains a social network with nodes as users and edges as the friendships [130].

- *Twitter*: This contains a social network where nodes represent users and edges represent the friendships [130].

- *Ciao*: This dataset contains a social network whose edges indicate the trust relationships among users and a set of user-product ratings with rich attribute information of product [151]. We extract 3,719 users who like more than 10 products and 4,612 products that are liked by these users. We use the product categories as the attributes. We consider ratings greater than or equal to 3 as interactions (i.e., likes) and obtain 105,900 user-product interactions.

With these datasets, we construct the scenarios of network alignment *S1-S3* and the recommendation scenario $S4$ for evaluations:

- *S1. Cora-1 vs. Cora-2*: Given the cora citation network, we generate two permuted networks $\mathcal{G}_1, \mathcal{G}_2$ and add noises by first inserting 10% edges to $\mathcal{G}_1$ and remove 15% edges from $\mathcal{G}_2$, and then adding 10% noises to $\mathbf{X}_1, \mathbf{X}_2$ (i.e., by randomly changing $0.1 \times \mathbf{1}'\mathbf{X}_0\mathbf{1}$ entries from 0 to 1). In this scenario, we aim to align the nodes in $\mathcal{G}_1, \mathcal{G}_2$. The permutation matrix is used as the ground-truth node correspondences.

- *S2. ACM vs. DBLP*: We aim to find the node correspondences across two co-author networks. There exist 6,325 common authors across two networks used as the ground-truth.

- *S3. Foursquare vs. Twitter*: In this scenario, we aim to align nodes in Foursquare and Twitter networks. There are 1,609 common users which are used as the ground-truth.

- *S4. Ciao users vs. product*: Different from the above scenarios, here we aim to predict the node associations between users and products indicating whether a user likes a product. To construct the product similarity network, similar to [133], we compute

Table 3.6: Data statistics.

Tasks	Networks	# of nodes	# of edges	# of attributes
Network Alignment	Cora-1	2,708	5,806	1,433
	Cora-2	2,708	4,547	1,433
	ACM	9,872	39,561	17
	DBLP	9,916	44,808	17
	Foursquare	5,313	54,233	0
	Twitter	5,120	130,575	0
Recommendation	Ciao-user	3,719	65,213	0
	Ciao-product	4,612	49,136	28

the similarities based on the embedding vectors of product reviews. The embedding vectors are learned by doc2vec [152]. Then we consider there exists an edge between two products if their similarity is larger than 0.5.

Besides, in *S1-S3*, we use 20% of the ground-truth as the training data (i.e., $\mathbf{L}$) and test on the rest of the ground-truth. In *S4*, we evaluate the performance in three sub-scenarios Ciao-r ($r \in \{0.2, 0.3, 0.5\}$) with different training ratios 20%, 30% and 50%, respectively.

Baseline methods. For network alignment, the baseline methods include: (1) FINAL-N [12], (2) FINAL-P [9], (3) REGAL [29] which is an embedding-based method for attributed networks, (4) IONE [8] and (5) CrossMNA [27] that are embedding-based methods without attributes. For fair comparisons, we modify FINAL-N and FINAL-P to semi-supervised setting using $\mathbf{L}$ as the prior alignment matrices. For one-class social recommendation, the baseline methods include (1) NGCF [153], (2) GraphRec [154] in which we use the BPR loss instead of the default mean square loss, (3) SamWalker [155], (4) wpZAN [149] that factorizes the node association matrix regularized by social network and product similarity network, and (5) BPR which is based on Bayesian personalized ranking loss [148].

Machine. The proposed model is implemented in Pytorch. We use one Nvidia GTX 1080 with 8G RAM as GPU.

Hyperparameters settings. We use Adam optimizer with a learning rate 0.005 to train the model. For network alignment, we set the margin size $\lambda = 0.1$, $\alpha = \beta = 1$, $\gamma = 10$ and the dimension of hidden representations in all layers to 256. As for the model architecture, in *S1*, we use $L = 2$, $n_1 = 2000$ and $n_2 = 1000$. In *S2*, due to the GPU memory limit, we use $L = 1$ and $n_1 = 5000$. In *S3*, we use $L = 2$, $n_1 = 5000$ and $n_2 = 2500$. For one-class social recommendation, we use the classic Bayesian personalized ranking loss to replace Eq. (3.88) and set $\alpha = \beta = 1$, $\gamma = 100$ and the dimensions of the representations to 128. The model architecture that we use is $L = 2$, $n_1 = 3000$ and $n_2 = 1500$. In both tasks, we set the negative sample size in Eq. (3.86) to 5 and that in the ranking loss Eq. (3.88) to 100. We

use the same embedding dimensions for embedding-based methods, and other parameters in the baseline methods are set to default.

Table 3.7: (Higher is better.) Effectiveness results on network alignment.

	Cora1-Cora2			ACM-DBLP			Foursquare-Twitter		
	Hits@10	Hits@30	Accuracy	Hits@10	Hits@30	Accuracy	Hits@10	Hits@30	Accuracy
NetTrans	**90.98%**	**97.51%**	**89.89%**	**84.09%**	**94.52%**	**58.21%**	**24.68%**	**34.58%**	**9.17%**
FINAL-N	88.73%	90.77%	87.58%	82.91%	90.71%	54.39%	24.09%	33.80%	8.47%
FINAL-P	62.28%	80.01%	54.34%	69.70%	83.12%	36.34%	24.09%	33.80%	8.47%
REGAL	60.90%	69.20%	46.26%	63.68%	71.80%	41.78%	0.15%	2.20%	0.11%
IONE	73.03%	79.92%	42.29%	58.93%	84.19%	33.00%	13.44%	28.17%	4.13%
CrossMNA	59.06%	68.62%	33.26%	42.54%	49.69%	21.04%	3.37%	14.79%	2.48%

Performance on Network Alignment. In this part, we compare our method NetTrans with the baseline methods in the scenarios *S1-S3*. We evaluate the effectiveness in terms of Hits@K and alignment accuracy. Given the testing node correspondence (e.g., u, v) across two networks, if $g_{\text{node}}(u, v)$ is among the highest top-K values within all the nodes in $\mathcal{G}_2$, then we say there is a *hit*. We count the number of hits, divided by the total number of testing node correspondences. Besides, the alignment accuracy measures the accuracy of the node one-to-one mappings obtained by using a greedy matching as a postprocess [9]. The results are summarized in Table 3.7. We have the following observations. First, our proposed method NetTrans outperforms both FINAL-N and FINAL-P. Specifically, it achieves an up to 6.5% improvement in Hits@30 and an up to 3% improvement in alignment accuracy, compared to FINAL-N. Note that both FINAL-N and FINAL-P can be viewed as the variants of the graph matching-based methods which, as mentioned before, are built upon the linearity/consistency assumptions. Second, our method achieves a better performance than other embedding-based network alignment methods (i.e., REGAL, IONE and CrossMNA). In particular, our method can achieve an at least 20% improvement in alignment accuracy on attributed networks (i.e., scenarios *S1, S2*). This demonstrates the improvements of our method compared to the embedding-based methods that suffer from the embedding space disparity. Third, we observe that even on the networks without node attributes, our proposed model still outperforms the baseline methods. Note that FINAL-N and FINAL-P have the same performance in *S3* because they are essentially equivalent without attributes.
Ablation study on the TransPool layer. To show the effectiveness of the proposed TransPool layer in identifying cross-network associations, we compare with two variants by replacing TransPool with the other pooling layers UNetPool [143] and SAGPool [144]. Since these two pooling layers originally do not learn the assignment matrices $\mathbf{P}_l$, we calculate the inner products between the node representations and the supernode representations in the l-th

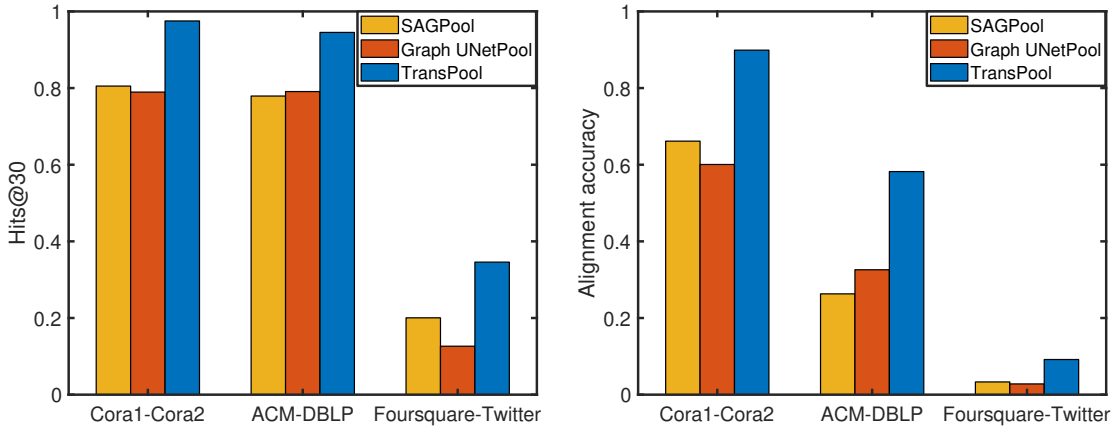


Figure 3.21: Ablation study on the pooling layer.

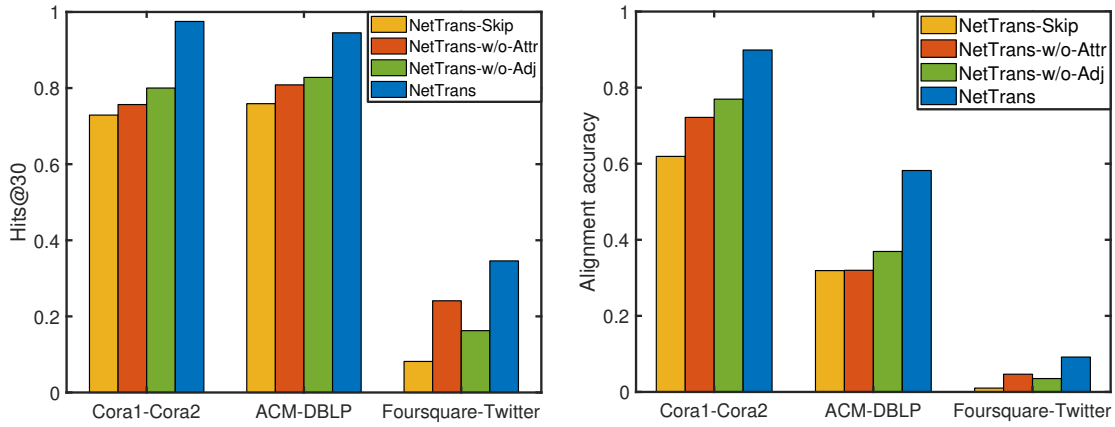


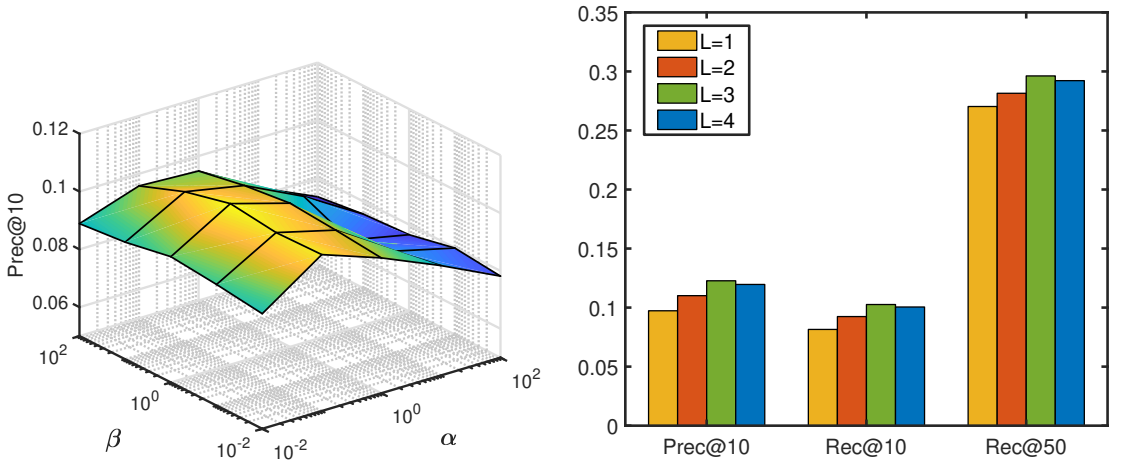
Figure 3.22: Ablation study on the unpooling layer.

pooling layer, followed by a softmax as $\mathbf{P}_l$. From Figure 3.21 we can see that using TransPool as the pooling layer significantly outperforms the other two variants. This implies the TransPool layer can learn better node representations and assignment matrices at different resolutions to help identify the cross-network node associations.

Ablation study on the TransUnPool layer. To show the effectiveness of the proposed unpooling layer, we compare with the different variants in learning the structure and node representations of the target network at different resolutions. These variants include: (1) NetTransSkip that directly uses the coarsened source network at the same resolution (i.e., $\mathbf{A}_1^{(L-k)} = \mathbf{A}_2^{(L-k)}$ and $\mathbf{X}_1^{(L-k)} = \mathbf{X}_2^{(L-k)}$), (2) NetTransw/o-Attr that only calibrates the structure (i.e., without calculating Eq. (3.80)), and (3) NetTransw/o-Adj that in contrast only calibrates the node representations (i.e., without calculating Eq. (3.81)). From Figure 3.22 we can see TransUnPool significantly outperforms other variants indicating the importance of the calibrations to earn the transformation across different networks.

Table 3.8: (Higher is better.) Effectiveness results on social recommendation.

	Ciao-0.2			Ciao-0.3			Ciao-0.5		
	Prec@10	Rec@10	Rec@50	Prec@10	Rec@10	Rec@50	Prec@10	Rec@10	Rec@50
NetTrans	**13.87%**	**11.08%**	**29.90%**	**11.01%**	**13.23%**	**28.15%**	**10.87%**	**12.43%**	**39.02%**
BPR	1.37%	0.6%	20.25%	1.38%	0.62%	20.18%	1.00%	0.37%	14.97%
wpZAN	11.99%	9.19%	20.77%	9.88%	10.33%	23.22%	9.85%	11.64%	26.04%
GraphRec	8.65%	6.62%	17.56%	8.42%	6.60%	18.07%	6.94%	6.63%	18.08%
SamWalker	4.94%	1.97%	5.98%	4.39%	2.07%	5.67%	2.48%	1.58%	4.05%
NGCF	2.77%	1.21%	3.26%	2.77%	1.48%	3.61%	3.17%	1.99%	4.77%



(a) Sensitivity of α and β.

(b) Sensitivity of L.

Figure 3.23: Sensitivity study on hyperparameters.

Performance on Recommendation. In addition to network alignment, we apply our proposed model to one-class social recommendation to predict whether users interact with certain products. In our experiments, we evaluate the performance by precision@K and recall@K. The results are summarized in Table 3.8. We have the following observations. First, our proposed method outperforms all the baseline methods. Specifically, our method achieves an at least 1% improvement in Precision@10 and an at least 5% improvement in Recall@50 compared to the best baseline method. Second, our method and wpZAN that is dual-regularized by both social and product networks outperform other baseline methods, which implies leveraging network structure, especially the product network, is indeed helpful.

Hyperparamter sensitivity study. We conduct the hyperparameter sensitivity studies for recommendation on the loss coefficients α, β and on the number of layers. From Figure 3.23 (a) we can see that our model is robust to different choices of α, β ranging from 0.01 to 100. In addition, from Figure 3.23 (b), we can observe that our model achieves the best performance with 3 encoder layers and 3 decoder layers. This demonstrates that learning hierarchical representations at different resolutions can lead to a better performance.

3.4 BALANCING CONSISTENCY AND DISPARITY IN NETWORK ALIGNMENT

In the age of big data, multiple networks emerge in many influential domains, such as social networks of different online social platforms and protein-protein interaction (PPI) networks of different species. Network alignment, which aims to uncover the node correspondences across networks (e.g., dashed lines in Figure 3.24 (a)), plays a crucial role in distilling values from multiple networks. Specifically, with the node correspondences, multiple networks can be integrated into a world-view network which may exhibit the patterns that are invisible if mining networks separately. For example, aligning proteins across PPI networks of different species facilitates to transfer knowledge from well-studied species to less-studied species and explore the evolutionary relationships [2]. In addition, by integrating multiple transaction networks, the complex fraud behaviors (e.g., money laundering) that are covert in each individual network can be brought to light.

Despite extensive research on network alignment, many traditional methods explicitly or implicitly assume the *alignment consistency* [3, 6, 9]. That is, the alignments of neighboring node pairs tend to be consistent with each other. For example, FINAL [9] explicitly formulates the alignment consistency as minimizing the differences between the alignment of two nodes and those of their corresponding neighbors. Alternatively, the objective can be interpreted as directly generating positive alignment pairs by *all* neighboring node pairs (e.g., (c, f) from the anchor link (b, y)) and minimizing the differences among them. However, optimizing this alignment consistency might result in the over-smoothness issue of the alignments within a local neighborhood and fail to distinguish correct alignments from misleading alignments.

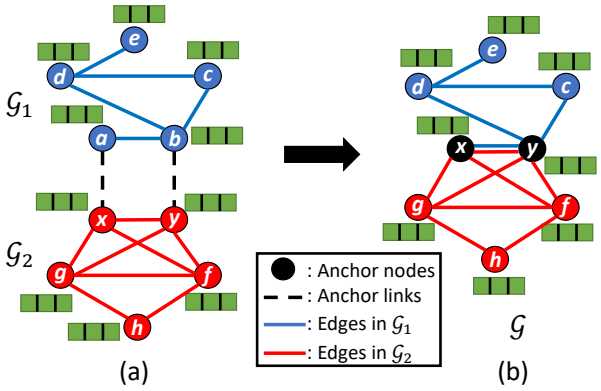


Figure 3.24: The world-view network.

On the other hand, embedding based methods, which infer node alignments by node embeddings, can to some extent incorporate the *alignment disparity* and ameliorate the over-smoothness issue by bringing in the negative alignment pairs. For example, some alignment methods (such as [8, 27]) sample negative alignment pairs by a degree-based sampling distribution and learn node embeddings that classify the sampled alignment pairs into the negative class. Others apply a uniform distribution to randomly sample negative alignment pairs used in the ranking loss [100, 101]. With negative sampling, the learned node embed-

dings can potentially make the alignments within the local neighborhood more separable (i.e., alignment disparity). More recently, negative sampling for *single* network embedding has been riveted and various sampling strategies are proposed under different or even competing design principles. For example, [156] advocates a positive correlation between negative and positive sampling distributions. That is, nodes with similar embeddings are more likely to be sampled such that *hard* negative samples are encouraged. Others exclusively favor a negative correlation to better preserve local proximity [157, 158].

Despite various negative sampling strategies, it still remains opaque how negative sampling should be designed to benefit network alignment. In other words, a more fundamental question is *what makes a 'good' negative pair in the context of network alignment?* Intuitively, good negative samples should distinguish the anchor links and the close node pairs that may mislead the alignment, while not violating the overall alignment consistency. In addition, these negative samples should inform the model to learn meaningful embeddings for network alignment. However, the aforementioned negative sampling strategies have their own limitations. To be specific, owing to its root in single network embedding, positive correlation based sampling [156] may lead to the false negative alignment pairs (e.g., (c, f) as the negative alignment pair of anchor link (b, y) in Figure 3.24 (a)) which results in incorrect alignments violating the alignment consistency. On the other side, using the pre-defined distributions [8, 100] and those that are negatively correlated to the positive sampling distribution [157, 158] may sample distant or dissimilar node pairs (e.g., (e, h)) that may not contribute much to learning meaningful node embeddings.

In this work, we strive to demystify the intrinsic relationships behind different competing designs for network alignment (i.e., alignment consistency vs. disparity, negative correlation vs. positive or neutral correlation), so that we can strike a good trade-off between them. We address this from both the model architecture and model training perspectives. First (*model design*), we theoretically prove that the alignments inferred by graph convolutional networks resemble the semi-supervised variant of the consistency based alignment method FINAL [9]. This inspires a specific graph convolutional network model that can preserve alignment consistency. Second (*model training*), we provide a lemma that implies the mean square error between the inner products of node embeddings learned by expected loss and empirical loss can be quantified by different sampling distributions. To reduce the error while making the aforementioned competing designs compatible with each other, we design a novel alignment scoring function which paves the way to the proposed sampling strategies. Armed with these components, we develop a novel semi-supervised method that accommodates both alignment consistency and alignment disparity. The main contributions are summarized as below.

- *Problem Definition.* To our best knowledge, we are the first to study the trade-off between the consistency and disparity in network alignment.
- *Method and Analysis.* We theoretically reveal the close connections between graph convolutional networks and consistency based alignment method and the intrinsic relationships behind the competing principles of sampling designs. Based on them, we develop a new semi-supervised network alignment method NeXtAlign.
- *Empirical Evaluations.* Extensive experiments validate significant improvements compared to the state-of-the-arts.

3.4.1 Problem Definition

Table 3.9 summarizes the main notations used in this work. We use bold uppercase letters for matrices (e.g., $\mathbf{L}$), bold lowercase letters for vectors (e.g., $\mathbf{a}$), lowercase letters (e.g., α) for scalars and uppercase calligraphic letters (e.g., $\mathcal{L}$) for sets. We denote the transpose by a superscript prime (e.g., $\mathbf{L}'$ as the transpose of $\mathbf{L}$).

Semi-Supervised Network Alignment. We denote input networks by $\mathcal{G}_1 = \{\mathcal{V}_1, \mathbf{A}_1, \mathbf{X}_1\}$ and $\mathcal{G}_2 = \{\mathcal{V}_2, \mathbf{A}_2, \mathbf{X}_2\}$ where $\mathcal{V}_1, \mathcal{V}_2$ represent the node sets, $\mathbf{A}_1, \mathbf{A}_2$ represent the adjacency matrices and $\mathbf{X}_1, \mathbf{X}_2$ represent the input node attributes of $\mathcal{G}_1, \mathcal{G}_2$ respectively. And we denote $n_1 = |\mathcal{V}_1|, n_2 = |\mathcal{V}_2|$ as the number of nodes in two networks and the input attribute matrices are of sizes $\mathbf{X}_1 \in \mathbb{R}^{d_0 \times n_1}, \mathbf{X}_2 \in \mathbb{R}^{d_0 \times n_2}$. In addition, anchor links are defined as the node pairs that are one-to-one mapped a priori. For example, given a set of anchor links $\mathcal{L} = \{(a, x) | a \in \mathcal{V}_1, x \in \mathcal{V}_2\}$, it indicates that node-a in $\mathcal{G}_1$ is aligned with node-x in $\mathcal{G}_2$ a priori. In this way, node-a is called as an anchor node in $\mathcal{G}_1$ and node-x is an anchor node in $\mathcal{G}_2$. We also define $\mathcal{L}_1 = \{a | \exists x \in \mathcal{V}_2, s.t., (a, x) \in \mathcal{L}\}$ as the set of anchor nodes in $\mathcal{G}_1$ and similarly $\mathcal{L}_2$ for $\mathcal{G}_2$. Accordingly, the sets of non-anchor nodes are denoted by $\bar{\mathcal{L}}_1 = \mathcal{V}_1 - \mathcal{L}_1$ and $\bar{\mathcal{L}}_2 = \mathcal{V}_2 - \mathcal{L}_2$. As for indexing nodes, we use b, y as the general indices to index all the nodes in $\mathcal{V}_1$ and $\mathcal{V}_2$. In addition, we use a, x to specifically index the anchor nodes in $\mathcal{L}_1, \mathcal{L}_2$, and use u, v to index non-anchor nodes.

Given the above notation definitions, our goal is to learn node embeddings and infer the alignment matrix $\mathbf{S} \in \mathbb{R}^{n_1 \times n_2}$. Formally, we define the semi-supervised network alignment.

Problem 3.5. Semi-Supervised Network Alignment.

Given: (1) undirected networks $\mathcal{G}_1 = \{\mathcal{V}_1, \mathbf{A}_1, \mathbf{X}_1\}$ and $\mathcal{G}_2 = \{\mathcal{V}_2, \mathbf{A}_2, \mathbf{X}_2\}$, and (2) a set of anchor links $\mathcal{L}$.

Output: alignment matrix $\mathbf{S}$ which indicates how likely nodes are aligned.

Table 3.9: Symbols and notations.

Symbols	Definition
$\mathcal{G}_1$, $\mathcal{G}_2$	input networks
$\mathcal{L}$	the set of anchor links across $\mathcal{G}_1, \mathcal{G}_2$
$\mathcal{L}_1, \mathcal{L}_2$	the sets of corresponding anchor nodes in $\mathcal{G}_1$ and $\mathcal{G}_2$
$\bar{\mathcal{L}}_1, \bar{\mathcal{L}}_2$	the sets of non-anchor nodes in $\mathcal{G}_1$ and $\mathcal{G}_2$
a, x	indices of anchor nodes in $\mathcal{G}_1$ and $\mathcal{G}_2$
u, v	indices of non-anchor nodes in $\mathcal{G}_1$ and $\mathcal{G}_2$
b, y	indices of all nodes in $\mathcal{G}_1$ and $\mathcal{G}_2$
$\mathbf{a}, \mathbf{x}$	column embedding vectors of node-a and node-x
$[\cdot\|\cdot]$	vertical concatenation of two column vectors
$\odot$	Hadamard product

Note that for networks without node attributes, we use the one-hot encoding of a node as its input node attributes. Given the fact that the nodes of an anchor link essentially represent the same entity, we can integrate the input networks $\mathcal{G}_1, \mathcal{G}_2$ into a world-view network $\mathcal{G}$ by merging two anchor nodes into a single node. As a result, the world-view network $\mathcal{G}$ has (1) non-anchor nodes in $\mathcal{G}_1, \mathcal{G}_2$ and unique anchor nodes as the nodes of $\mathcal{G}$, and (2) all edges of $\mathcal{G}_1, \mathcal{G}_2$ co-exist in $\mathcal{G}$ (shown in Figure 3.24 (b)). By learning node embeddings in this world-view network $\mathcal{G}$, we can naturally share the unique embedding across two corresponding anchor nodes, i.e., $\mathbf{a} = \mathbf{x},\ \forall (a, x) \in \mathcal{L}$. In this study, we use $\mathbf{a}$ and $\mathbf{x}$ interchangeably.

Preliminaries. In many prior network alignment methods, a widely used assumption is the *alignment consistency* assumption that neighboring node pairs tend to have consistent alignments [3, 5, 9]. The prior method FINAL [9] formulates this assumption as

$$\min_{\mathbf{S}} \sum_{a,b,x,y} \left[\frac{\mathbf{S}(a,x)}{\sqrt{|\mathcal{N}_1(a)||\mathcal{N}_2(x)|}} - \frac{\mathbf{S}(b,y)}{\sqrt{|\mathcal{N}_1(b)||\mathcal{N}_2(y)|}} \right]^2 \mathbf{A}_1(a,b)\mathbf{A}_2(x,y) \tag{3.91}$$

where $\mathcal{N}_1(a), \mathcal{N}_2(x)$ denote the neighbors of node-a in $\mathcal{G}_1$ and node-x in $\mathcal{G}_2$. Besides, we have $\mathbf{A}_1 = \mathbf{A}_1', \mathbf{A}_2 = \mathbf{A}_2'$. Here, Eq. (3.91) encourages a small difference between $\mathbf{S}(a,x)$ and $\mathbf{S}(b,y)$ if node-b and node-y are close neighbors of node-a and node-x. We can interpret Eq. (3.91) from another angle. That is, given an anchor link (a,x) with a large $\mathbf{S}(a,x)$, it encourages $\mathbf{S}(b,y)$ to be consistent with $\mathbf{S}(a,x)$ (i.e., large $\mathbf{S}(b,y)$), which means Eq. (3.91) naturally considers all the neighboring node pairs as the positive alignment pairs of (a,x). To solve Eq. (3.91), a fixed-point update in the t-th iteration is computed as

$$\mathbf{S}^t = \tilde{\mathbf{A}}_1 \mathbf{S}^{t-1} \tilde{\mathbf{A}}_2' \tag{3.92}$$

where $\tilde{\mathbf{A}}_1, \tilde{\mathbf{A}}_2$ are the symmetric normalization of $\mathbf{A}_1, \mathbf{A}_2$.

In the semi-supervised setting, the supervision (i.e., anchor links) can be used as a regularization upon Eq. (3.92), which leads to

$$\mathbf{S}^t = \alpha \tilde{\mathbf{A}}_1 \mathbf{S}^{t-1} \tilde{\mathbf{A}}_2' + (1-\alpha)\mathbf{L} \tag{3.93}$$

where α controls the importance of the alignment consistency. $\mathbf{L}(a,:) = 0$ and $\mathbf{L}(:,x) = 0$ except $\mathbf{L}(a,x) = 1$ if and only if $(a,x) \in \mathcal{L}$.

3.4.2 The NextAlign Model

In this part, we present NeXtAlign, a semi-supervised network alignment method. We start by giving a model overview. Then, we introduce the model design for embedding learning, followed by the proposed negative sampling for model training.

Model Overview and Key Ideas. The core challenge that we aim to address is to design and train the model to strike a balance between the alignment consistency and disparity. To design the model that learns node embeddings while encouraging alignment consistency, we first prove that the alignments inferred by the node embeddings by a specific message passing without parameters (Eq. (3.96)) resemble the semi-supervised FINAL [9]. The key idea of the proof is to conduct a rank-$|\mathcal{L}|$ decomposition on matrix $\mathbf{L}$ used in Eq. (3.93) and use the decomposed matrices as the input node embeddings. Intuitively, by viewing the anchor nodes as the landmarks in the $|\mathcal{L}|$-dimensional Euclidean space, this message passing can be interpreted as to determine the relative positions for all nodes w.r.t. the anchor nodes. Next, based on its capability of capturing alignment consistency, we propose the parameterized counterpart of this message passing in Eq. (3.103). We name it as the RelGCN layer which is then used to calibrate the relative positions of nodes calculated by Eq. (3.96). The final node embeddings are obtained by applying a linear layer on these position vectors. The overall architecture is shown in Figure 3.25.

In terms of model training, the key idea of achieving the trade-off is by different sampling distributions. The intuitions behind it are as follows. Given an anchor link $(a,x) \in \mathcal{L}$, if (b,y) is sampled as the positive alignment pair, it encourages the consistency between node pairs (b,y) and (a,x). In contrast, if (b,y) is sampled as the negative alignment pair, the alignment disparity is favored between them. Besides, to preserve the local proximity within the same network, we also sample positive context pairs and negative context pairs. To design these sampling distributions, we first quantify the mean square error between the inner products

of node embeddings learned by minimizing the expected loss and empirical loss. Based on this, to make the inner products of the high-probability node pairs to be estimated more accurately while satisfying different and even competing design principles, we compose a novel alignment scoring function that reflects multiple aspects of node embeddings.

Model Design. We first connect the fixed-point update of FINAL (i.e., Eq. (3.91)) to the vanilla GCN [122]. Suppose the alignment is computed by $\mathbf{S}(a,x) = \mathbf{a}'\mathbf{x}$, then we have

$$\begin{aligned}(\mathbf{a}^t)'\mathbf{x}^t = \mathbf{S}^t(a,x) &= \tilde{\mathbf{A}}_1(a,:)\mathbf{S}^{t-1}\tilde{\mathbf{A}}_2(:,x) \\ &= \sum_{b\in\mathcal{N}_1(a)}\sum_{y\in\mathcal{N}_2(x)} \frac{(\mathbf{b}^{t-1})'\mathbf{y}^{t-1}}{\sqrt{|\mathcal{N}_1(a)||\mathcal{N}_1(b)||\mathcal{N}_2(x)||\mathcal{N}_2(y)|}} \\ &= \sum_{b\in\mathcal{N}_1(a)} \frac{(\mathbf{b}^{t-1})'}{\sqrt{|\mathcal{N}_1(a)||\mathcal{N}_1(b)|}} \sum_{y\in\mathcal{N}_2(x)} \frac{\mathbf{y}^{t-1}}{\sqrt{|\mathcal{N}_2(x)||\mathcal{N}_2(y)|}}\end{aligned} \tag{3.94}$$

where $\mathbf{b}^{t-1}$ represents the node embedding of node-b in the $(t-1)$-th iteration/layer. As we can see, the t-th iteration of computing the alignment $\mathbf{S}^t(a,x)$ is equivalent to updating the node embeddings by applying the vanilla GCN without parameters

$$\mathbf{a}^t = \sum_{b\in\mathcal{N}_1(a)} \frac{\mathbf{b}^{t-1}}{\sqrt{|\mathcal{N}_1(a)||\mathcal{N}_1(b)|}}, \quad \mathbf{x}^t = \sum_{y\in\mathcal{N}_2(x)} \frac{\mathbf{y}^{t-1}}{\sqrt{|\mathcal{N}_2(x)||\mathcal{N}_2(y)|}} \tag{3.95}$$

followed by the inner product. In addition, due to the over-smoothness issue of GCNs [159], the node alignments inferred by the node embeddings above could also suffer from over-smoothness, which might hamper the performance.

In the semi-supervised setting where anchor links are available, we design the following message passing without parameters

$$\begin{aligned}\mathbf{u}^t &= \sqrt{\alpha}\sum_{b\in\mathcal{N}_1(u)} \frac{\mathbf{b}^{t-1}}{\sqrt{|\mathcal{N}_1(u)||\mathcal{N}_1(b)|}} + \sqrt{1-\alpha}\mathbf{u}^{t-1} \\ \mathbf{v}^t &= \sqrt{\alpha}\sum_{y\in\mathcal{N}_2(v)} \frac{\mathbf{y}^{t-1}}{\sqrt{|\mathcal{N}_2(v)||\mathcal{N}_2(y)|}} + \sqrt{1-\alpha}\mathbf{v}^{t-1} \\ \mathbf{a}^t = \mathbf{x}^t &= \sqrt{\alpha}\sum_{b\in\mathcal{N}_1(a)} \frac{\mathbf{b}^{t-1}}{\sqrt{|\mathcal{N}_1(a)||\mathcal{N}_1(b)|}} + \sqrt{1-\alpha}\mathbf{x}^{t-1} \\ &\quad + \sqrt{\alpha}\sum_{y\in\mathcal{N}_2(x)} \frac{\mathbf{y}^{t-1}}{\sqrt{|\mathcal{N}_2(x)||\mathcal{N}_2(y)|}}\end{aligned} \tag{3.96}$$

where node-u, node-v are non-anchor nodes and a, x are anchor nodes. As shown in Lemma 3.7, the alignments inferred by embeddings in Eq. (3.96) resemble Eq. (3.93).

Lemma 3.7. Suppose the initial non-anchor node embeddings are $\mathbf{u}^0 = \mathbf{v}^0 = \mathbf{0}$ and those of the anchor nodes are $\mathbf{a}^0 = \mathbf{x}^0 = \mathbf{e}_i \in \mathbb{R}^{|\mathcal{L}|}$ where (a, x) is the i-th anchor link, $\mathbf{e}_i(i) = 1$ and $\mathbf{e}_i(j) = 0,\ \forall j \neq i$. Then by updating Eq. (3.96) once, the alignments computations are equivalent to Eq. (3.93) up to additional intra-network proximity and scaling terms.

Proof. Given $|\mathcal{L}| = L$ anchor links, we can conduct a rank-L decomposition upon $\mathbf{L}$ without errors into $\mathbf{L} = \mathbf{L}_1'\mathbf{L}_2$. Since $\mathbf{L}(a, x) = 1$ if $(a, x) \in \mathcal{L}$, we have $\mathbf{L}_1(:, a) = \mathbf{a}^0 = \mathbf{e}_i$ and $\mathbf{L}_2(:, x) = \mathbf{x}^0 = \mathbf{e}_i$. For non-anchor nodes, we have $\mathbf{L}_1(:, u) = \mathbf{u}^0 = \mathbf{0}$, $\mathbf{L}_2(:, v) = \mathbf{v}^0 = \mathbf{0}$. After initializing embeddings as $\mathbf{L}_1, \mathbf{L}_2$, the alignments are computed by the inner products among the updated embeddings with Eq. (3.96).

$$\mathbf{S}(u, v) = \alpha \sum_{b \in \mathcal{N}_1(u)} \frac{(\mathbf{b}^0)'}{\sqrt{|\mathcal{N}_1(u)||\mathcal{N}_1(b)|}} \sum_{y \in \mathcal{N}_2(v)} \frac{\mathbf{y}^0}{\sqrt{|\mathcal{N}_2(v)||\mathcal{N}_2(y)|}} \tag{3.97}$$

$$\begin{aligned}\mathbf{S}(u, x) = {} & \alpha \sum_{b \in \mathcal{N}_1(u)} \frac{(\mathbf{b}^0)'}{\sqrt{|\mathcal{N}_1(u)||\mathcal{N}_1(b)|}} \sum_{y \in \mathcal{N}_2(x)} \frac{\mathbf{y}^0}{\sqrt{|\mathcal{N}_2(x)||\mathcal{N}_2(y)|}} \\ & + \alpha \sum_{b \in \mathcal{N}_1(u)} \frac{(\mathbf{b}^0)'}{\sqrt{|\mathcal{N}_1(u)||\mathcal{N}_1(b)|}} \sum_{c \in \mathcal{N}_1(a)} \frac{\mathbf{c}^0}{\sqrt{|\mathcal{N}_1(a)||\mathcal{N}_1(c)|}} \\ & + \sqrt{\alpha(1-\alpha)} \sum_{b \in \mathcal{N}_1(u)} \frac{(\mathbf{b}^0)'}{\sqrt{|\mathcal{N}_1(u)||\mathcal{N}_1(b)|}} \mathbf{x}^0\end{aligned} \tag{3.98}$$

$$\begin{aligned}\mathbf{S}(a, x) = {} & 2\alpha \sum_{b \in \mathcal{N}_1(a)} \frac{(\mathbf{b}^0)'}{\sqrt{|\mathcal{N}_1(a)||\mathcal{N}_1(b)|}} \sum_{y \in \mathcal{N}_2(x)} \frac{\mathbf{y}^0}{\sqrt{|\mathcal{N}_2(x)||\mathcal{N}_2(y)|}} \\ & + \frac{\alpha}{|\mathcal{N}_1(a)|} \sum_{b \in \{\mathcal{N}_1(a) \cap \mathcal{L}_1\}} \frac{1}{|\mathcal{N}_1(b)|} + \frac{\alpha}{|\mathcal{N}_2(x)|} \sum_{y \in \{\mathcal{N}_2(x) \cap \mathcal{L}_2\}} \frac{1}{|\mathcal{N}_2(x)|} + (1 - \alpha)\end{aligned} \tag{3.99}$$

Note $\mathbf{S}(a, v)$ is omitted as it is similar to $\mathbf{S}(u, x)$. We denote

$$\mathbf{S}_1(u, a) = \left[\sum_{b \in \mathcal{N}_1(u)} \frac{\mathbf{b}^0}{\sqrt{|\mathcal{N}_1(u)||\mathcal{N}_1(b)|}}\right]' \left[\sum_{c \in \mathcal{N}_1(a)} \frac{\mathbf{c}^0}{\sqrt{|\mathcal{N}_1(a)||\mathcal{N}_1(c)|}}\right] \tag{3.100}$$

$$\mathbf{S}_2(x, v) = \left[\sum_{y \in \mathcal{N}_2(x)} \frac{\mathbf{y}^0}{\sqrt{|\mathcal{N}_2(y)||\mathcal{N}_2(x)|}}\right]' \left[\sum_{z \in \mathcal{N}_2(v)} \frac{\mathbf{z}^0}{\sqrt{|\mathcal{N}_2(z)||\mathcal{N}_1(v)|}}\right] \tag{3.101}$$

which measure the weighted number of common neighboring anchor nodes. Recall Eq.

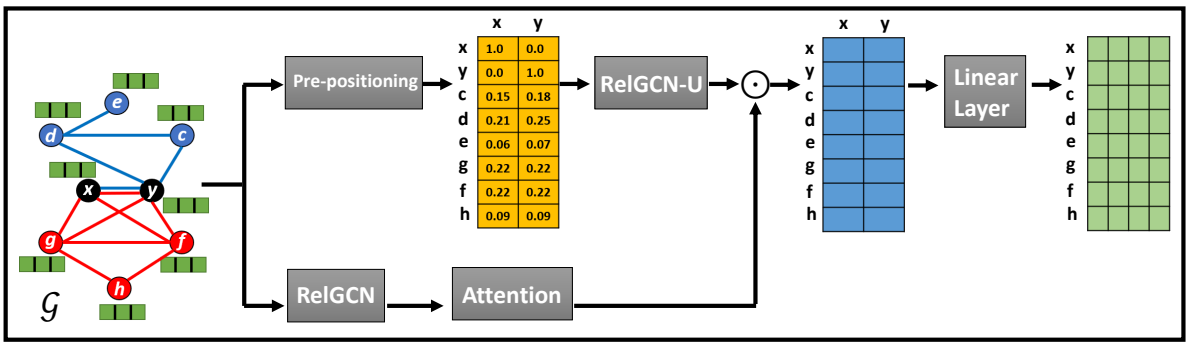


Figure 3.25: Overall architecture of NeXtAlign.

(3.94), $\mathbf{L}(u,v) = \mathbf{L}(u,x) = 0$ and $\mathbf{L}(a,x) = 1$, then we have

$$\begin{aligned}
\mathbf{S}(u,v) &= \alpha\tilde{\mathbf{A}}_1(u,:)\mathbf{L}\tilde{\mathbf{A}}_2(:,v) + (1-\alpha)\mathbf{L}(u,v) \\
\mathbf{S}(u,x) &= \alpha\tilde{\mathbf{A}}_1(u,:)\mathbf{L}\tilde{\mathbf{A}}_2(:,x) + (1-\alpha)\mathbf{L}(u,x) + \alpha\mathbf{S}_1(u,a) \\
&\quad + \sqrt{\alpha(1-\alpha)}\frac{\mathbf{A}_1(u,a)}{\sqrt{|\mathcal{N}_1(u)||\mathcal{N}_1(a)|}} \\
\mathbf{S}(a,x) &= 2\alpha\tilde{\mathbf{A}}_1(a,:)\mathbf{L}\tilde{\mathbf{A}}_2(:,x) + \alpha(\mathbf{S}_1(a,a) + \mathbf{S}_2(x,x)) + (1-\alpha)\mathbf{L}(a,x)
\end{aligned} \tag{3.102}$$

As we can see, the alignments based on the embeddings learned by Eq. (3.96) in the *first* iteration are equivalent to the semi-supervised FINAL (i.e., Eq. (3.93)) with $\mathbf{S}^0 = \mathbf{L}$ except the additional intra-network proximity (e.g., $\mathbf{S}_1, \mathbf{S}_2$) and scaling terms. QED.

This lemma implies that we can design a special relational graph convolutional network (RelGCN) to encode the alignment consistency. We formulate the t-th RelGCN layer as

$$\begin{aligned}
\mathbf{u}^t &= \sqrt{\alpha}\sum_{b\in\mathcal{N}_1(u)}\frac{\mathbf{W}_1^t\mathbf{b}^{t-1}}{\sqrt{|\mathcal{N}_1(u)||\mathcal{N}_1(b)|}} + \sqrt{1-\alpha}\mathbf{W}_0^t\mathbf{u}^{t-1} \\
\mathbf{v}^t &= \sqrt{\alpha}\sum_{y\in\mathcal{N}_2(v)}\frac{\mathbf{W}_2^t\mathbf{y}^{t-1}}{\sqrt{|\mathcal{N}_2(v)||\mathcal{N}_2(y)|}} + \sqrt{1-\alpha}\mathbf{W}_0^t\mathbf{v}^{t-1} \\
\mathbf{a}^t = \mathbf{x}^t &= \sqrt{\alpha}\sum_{b\in\mathcal{N}_1(a)}\frac{\mathbf{W}_1^t\mathbf{b}^{t-1}}{\sqrt{|\mathcal{N}_1(a)||\mathcal{N}_1(b)|}} + \sqrt{1-\alpha}\mathbf{W}_0^t\mathbf{x}^{t-1} \\
&\quad + \sqrt{\alpha}\sum_{y\in\mathcal{N}_2(x)}\frac{\mathbf{W}_2^t\mathbf{y}^{t-1}}{\sqrt{|\mathcal{N}_2(x)||\mathcal{N}_2(y)|}}
\end{aligned} \tag{3.103}$$

where $\mathbf{W}_0^t, \mathbf{W}_1^t, \mathbf{W}_2^t \in \mathbb{R}^{d_t \times d_{t-1}}$ and d_t denotes the embedding dimension in the t-th layer of RelGCN. We name the RelGCN layer without parameters (i.e., Eq. (3.96)) as RelGCN-U.

We then design the whole model architecture shown in Figure 3.25. The key idea is to

leverage RelGCNs to learn node embeddings that describe the *relative 'positions'* of the nodes w.r.t. the anchor nodes [32], followed by a linear layer to learn the final output embeddings. In particular, given the prior alignment matrix $\mathbf{L}$, we first decompose it into two rank-L matrices by $\mathbf{L} = \mathbf{L}_1'\mathbf{L}_2$ as the initial embeddings. Then we feed them into RelGCN-U to incorporate the alignment consistency. For the nodes that are not adjacent to any anchor nodes, we apply random walk with restart [107] to measure the proximities of non-anchor nodes w.r.t. the anchor nodes as the initial relative positions

$$\mathbf{r}_{i1} = (1-p)\hat{\mathbf{A}}_1\mathbf{r}_{i1} + p\hat{\mathbf{e}}_{i1}, \qquad \mathbf{r}_{i2} = (1-p)\hat{\mathbf{A}}_2\mathbf{r}_{i2} + p\hat{\mathbf{e}}_{i2} \tag{3.104}$$

where the restart probability p is set to 0.85 following the classic choice, and $\hat{\mathbf{A}}_1, \hat{\mathbf{A}}_2$ are the normalized matrices of $\mathbf{A}_1, \mathbf{A}_2$. The restart vectors $\hat{\mathbf{e}}_{i1} \in \mathbb{R}^{n_1}, \hat{\mathbf{e}}_{i2} \in \mathbb{R}^{n_2}$ only have one nonzero value at $\hat{\mathbf{e}}_{i1}(a) = 1$ and $\hat{\mathbf{e}}_{i2}(x) = 1$. After achieving the stationary distributions, we set $\mathbf{u}^0 = [\mathbf{r}_{11}(u), \mathbf{r}_{21}(u), \cdots, \mathbf{r}_{L1}(u)]'$ for non-anchor node-u in $\mathcal{G}_1$, and similarly for non-anchor nodes v in $\mathcal{G}_2$. We denote the output embeddings by RelGCN-U by, say, $\mathbf{a}^1, \mathbf{x}^1, \mathbf{u}^1, \mathbf{v}^1$.

However, the alignment scores among the close neighborhood might not distinguish the precise node alignments due to the over-smoothness. To mitigate the issue, we additionally leverage attention coefficients to rescale the relative positions computed by

$$\begin{aligned}
\hat{\mathbf{u}} &= \sqrt{\alpha} \sum_{b \in \mathcal{N}_1(u)} \frac{\mathbf{W}_1\mathbf{X}_1(:,b)}{\sqrt{|\mathcal{N}_1(u)||\mathcal{N}_1(b)|}} + \sqrt{1-\alpha}\mathbf{W}_0\mathbf{X}_1(:,u) \\
\hat{\mathbf{a}} = \hat{\mathbf{x}} &= \sqrt{\alpha} \sum_{b \in \mathcal{N}_1(a)} \frac{\mathbf{W}_1\mathbf{X}_1(:,b)}{\sqrt{|\mathcal{N}_1(a)||\mathcal{N}_1(b)|}} + \sqrt{1-\alpha}\mathbf{W}_0\mathbf{X}_1(:,a) \\
&\quad + \sqrt{\alpha} \sum_{y \in \mathcal{N}_2(x)} \frac{\mathbf{W}_2\mathbf{X}_2(:,y)}{\sqrt{|\mathcal{N}_2(x)||\mathcal{N}_2(y)|}} \\
c_{ua} &= \frac{\exp\left(\mathbf{w}_c'[\hat{\mathbf{u}} \| \hat{\mathbf{a}}]\right)}{\sum_{b \in \mathcal{L}_1} \exp\left(\mathbf{w}_c'[\hat{\mathbf{u}} \| \hat{\mathbf{b}}]\right)}
\end{aligned} \tag{3.105}$$

and similarly for c_{vx} where $\mathbf{w}_c$ is a parameter vector. We scale the relative positions by

$$\tilde{\mathbf{u}} = \hat{\mathbf{u}} \odot \mathbf{c}_u, \quad \tilde{\mathbf{v}} = \hat{\mathbf{v}} \odot \mathbf{c}_v, \quad \tilde{\mathbf{a}} = \tilde{\mathbf{x}} = \hat{\mathbf{x}} \odot \mathbf{c}_x \tag{3.106}$$

where $\mathbf{c}_u(i) = c_{ua}, \mathbf{c}_v(i) = c_{vx}$. To learn the embeddings, we apply a simple linear layer

$$\mathbf{u} = \mathbf{W}\tilde{\mathbf{u}}, \quad \mathbf{v} = \mathbf{W}\tilde{\mathbf{v}}, \quad \mathbf{a} = \mathbf{x} = \mathbf{W}\tilde{\mathbf{x}} \tag{3.107}$$

where $\mathbf{W} \in \mathbb{R}^{d \times |\mathcal{L}|}$ is the corresponding weight parameter matrix.

Model Training. We first consider the following loss functions on anchor links that capture both intra-network information by J_a, J_x and inter-network information by J_{ax}.

$$J_a = -\sum_b [p_d(b|a)\log\sigma(\mathbf{b}'\mathbf{a}) + kp_n(b|a)\log\sigma(-\mathbf{b}'\mathbf{a})] \tag{3.108}$$

$$J_x = -\sum_y [p_d(y|x)\log\sigma(\mathbf{y}'\mathbf{x}) + kp_n(y|x)\log\sigma(-\mathbf{y}'\mathbf{x})] \tag{3.109}$$

$$J_{ax} = -\sum_b [p_{dc}(b|x)\log\sigma(\mathbf{b}'\mathbf{x}) + kp_{nc}(b|x)\log\sigma(-\mathbf{b}'\mathbf{x})] \tag{3.110}$$

$$-\sum_y [p_{dc}(y|a)\log\sigma(\mathbf{y}'\mathbf{a}) + kp_{nc}(y|a)\log\sigma(-\mathbf{y}'\mathbf{a})]$$

$$J = \sum_{(a,x)\in\mathcal{L}} J_{(a,x)} = \sum_{(a,x)\in\mathcal{L}} J_a + J_x + J_{ax} \tag{3.111}$$

where $\sigma(\cdot)$ is the sigmoid function. The probability distributions p_d, p_n sample the positive and negative context node pairs within the same network respectively, while p_{dc}, p_{nc} sample positive and negative alignment pairs across different networks. Note that in the above loss functions, we assume for simplicity that the probabilities for the same goal are calculated by the same function (e.g., $p_d(\cdot|a)$ in $\mathcal{G}_1$ vs. $p_d(\cdot|x)$ in $\mathcal{G}_2$ using the same function but different inputs) and the numbers of negative samples by p_n, p_{nc} are same (i.e., k). Since $\mathbf{a} = \mathbf{x},\ \forall(a,x) \in \mathcal{L}$, we can rewrite the loss related to the anchor link (a,x) as below.

$$J_{(a,x)} = -\sum_b [[p_d(b|a) + p_{dc}(b|x)]\log\sigma(\mathbf{b}'\mathbf{x}) + k[p_n(b|a) + p_{nc}(b|x)]\log\sigma(-\mathbf{b}'\mathbf{x})] \tag{3.112}$$

$$-\sum_y [[p_d(y|x) + p_{dc}(y|a)]\log\sigma(\mathbf{y}'\mathbf{x}) + k[p_n(y|x) + p_{nc}(y|a)]\log\sigma(-\mathbf{y}'\mathbf{x})] \tag{3.113}$$

For the anchor link (a,x), we derive the conditions of the optimal node embeddings [156].

Lemma 3.8. Given an anchor link (a,x), the optimal embeddings that minimize $J_{(a,x)}$ satisfy for non-anchor nodes $b \in \bar{\mathcal{L}}_1, y \in \bar{\mathcal{L}}_2$,

$$\mathbf{b}'\mathbf{x} = -\log\frac{kp_n(b|a) + kp_{nc}(b|x)}{p_d(b|a) + p_{dc}(b|x)} \tag{3.114}$$

$$\mathbf{y}'\mathbf{x} = -\log\frac{kp_n(y|x) + kp_{nc}(y|a)}{p_d(y|x) + p_{dc}(y|a)} \tag{3.115}$$

and for anchor nodes such that $(b,y) \in \mathcal{L}$,

$$\mathbf{b}'\mathbf{x} = \mathbf{y}'\mathbf{x} = -\log\frac{k[p_n(b|a) + p_{nc}(b|x) + p_n(y|x) + p_{nc}(y|a)]}{p_d(b|a) + p_{dc}(b|x) + p_d(y|x) + p_{nc}(y|a)} \tag{3.116}$$

Proof. For node-b in $\mathcal{G}_1$ and node-y in $\mathcal{G}_2$ such that $(b, y) \notin \mathcal{L}$, the main idea is to first prove that the loss functions Eq. (3.112) and Eq. (3.113) can be minimized separately by satisfying the conditions Eq. (3.114) and Eq. (3.115), and then prove these two conditions can co-occur. We first define two Bernoulli distributions $P_{b,(a,x)}(z = 1) = \frac{p_d(b|a)+p_{dc}(b|x)}{p_d(b|a)+p_{dc}(b|x)+kp_n(b|a)+kp_{nc}(b|x)}$ and $Q_{b,(a,x)}(z = 1) = \sigma(\mathbf{b}'\mathbf{a}) = \sigma(\mathbf{b}'\mathbf{x})$. Then the term of node-$b$ in Eq. (3.112) is

$$O_b = [p_d(b|a) + p_{dc}(b|x) + kp_n(b|a) + kp_{nc}(b|x)]H(P_{b,(a,x)}, Q_{b,(a,x)}) \tag{3.117}$$

where $H(\cdot, \cdot)$ is the cross-entropy between two distributions. According to Gibbs Inequality, the minimum can be achieved when $P_{b,(a,x)} = Q_{b,(a,x)},\ \forall b \in \mathcal{V}_1 - \{b|(b, y) \notin \mathcal{L}\}$. This implies

$$\mathbf{b}'\mathbf{x} = -\log \frac{kp_n(b|a) + kp_{nc}(b|x)}{p_d(b|a) + p_{dc}(b|x)} \tag{3.118}$$

Similarly, we can show the loss Eq. (3.113) is minimized when

$$\mathbf{y}'\mathbf{x} = -\log \frac{kp_n(y|x) + kp_{nc}(y|a)}{p_d(y|x) + p_{dc}(y|a)} \tag{3.119}$$

Since $(b, y) \notin \mathcal{L}$, it is easy to see that at least one of $\mathbf{b}$ and $\mathbf{y}$ could be an arbitrary vector as long as it satisfies the above condition. Next, for two nodes such that $(b, y) \in \mathcal{L}$ and consequently $\mathbf{b} = \mathbf{y}$, the corresponding term in Eq. (3.112) is equivalent to

$$\begin{aligned} O_b = &- [p_d(b|a) + p_{dc}(b|x) + p_d(y|x) + p_{dc}(y|a)] \log \sigma(\mathbf{b}'\mathbf{x}) \\ &- [p_n(b|a) + p_{nc}(b|x) + p_n(y|x) + p_{nc}(y|a)] \log \sigma(-\mathbf{b}'\mathbf{x}) \end{aligned} \tag{3.120}$$

Similarly, we can derive the condition for $(b, y) \in \mathcal{L}$ as

$$\mathbf{b}'\mathbf{x} = \mathbf{y}'\mathbf{x} = -\log \frac{k[p_n(b|a) + p_{nc}(b|x) + p_n(y|x) + p_{nc}(b|a)]}{p_d(b|a) + p_{dc}(b|x) + p_d(y|x) + p_{nc}(y|a)} \tag{3.121}$$

This completes the proof of Lemma 3.8. QED.

However, the above lemma requires sufficient (probably infinite) sampled node pairs. In this way, we further consider to minimize the empirical risk for an anchor link (a, x) as

$$\begin{aligned} J^B_{(a,x)} = &-\frac{1}{B} \sum_{i_1,i_2,j_1,j_2} \log \sigma(\mathbf{b}'_{i_1}\mathbf{x}) + \log \sigma(\mathbf{b}'_{i_2}\mathbf{x}) + \log \sigma(\mathbf{y}'_{j_1}\mathbf{x}) + \log \sigma(\mathbf{y}'_{j_2}\mathbf{x}) \\ &- \frac{1}{B} \sum_{i_3,i_4,j_3,j_4} \left[\log \sigma(-\mathbf{b}'_{i_3}\mathbf{x}) + \log \sigma(-\mathbf{b}'_{i_4}\mathbf{x}) + \log \sigma(-\mathbf{y}'_{j_3}\mathbf{x}) + \log \sigma(-\mathbf{y}'_{j_4}\mathbf{x})\right] \end{aligned} \tag{3.122}$$

where B is the number of positive samples and accordingly kB is the size of negative samples. In addition, (1) b_{i_1}, y_{j_1} are sampled from $p_d(\cdot|a)$, $p_d(\cdot|x)$, (2) b_{i_2}, y_{j_2} are sampled from $p_{dc}(\cdot|x)$, $p_{dc}(\cdot|a)$, (3) b_{i_3}, y_{j_3} are sampled from $p_n(\cdot|a)$, $p_n(\cdot|x)$, and (4) b_{i_4}, y_{j_4} are sampled from $p_{nc}(\cdot|x)$, $p_{nc}(\cdot|a)$ respectively. Furthermore, by defining $\boldsymbol{\theta} = [\mathbf{b}_1'\mathbf{x}, \cdots, \mathbf{b}_{n_1}'\mathbf{x}, \mathbf{y}_1'\mathbf{x}, \cdots, \mathbf{y}_{n_2}'\mathbf{x}]$, we denote $\boldsymbol{\theta}^*$ as the optimal solution to $J_{(a,x)}$ and $\boldsymbol{\theta}^B$ similarly for the empirical risk $J^B_{(a,x)}$. Then we can derive the mean square error in the following lemma.

Lemma 3.9. Denote $\Delta\boldsymbol{\theta}_b = \boldsymbol{\theta}_b^B - \boldsymbol{\theta}_b^*$ and $\Delta\boldsymbol{\theta}_y = \boldsymbol{\theta}_y^B - \boldsymbol{\theta}_y^*$. The mean square errors for nodes $b \in \bar{\mathcal{L}}_1$ and $y \in \bar{\mathcal{L}}_2$ can be formulated by

$$\begin{aligned}\mathbb{E}\left[\Delta\boldsymbol{\theta}_b^2\right] &= \frac{1}{B}\left[\frac{1}{p_d(b|a)+p_{dc}(b|x)} + \frac{1}{kp_n(b|a)+kp_{nc}(b|x)} - C\right] \\ \mathbb{E}[\Delta\boldsymbol{\theta}_y^2] &= \frac{1}{B}\left[\frac{1}{p_d(y|x)+p_{dc}(y|a)} + \frac{1}{kp_n(y|x)+kp_{nc}(y|a)} - C\right]\end{aligned} \tag{3.123}$$

For nodes $b \in \mathcal{L}_1$ and $y \in \mathcal{L}_2$, by denoting $C = 1 + \frac{1}{k}$, the mean square error is computed by

$$\mathbb{E}\left[\Delta\boldsymbol{\theta}_b^2\right] = \mathbb{E}[\Delta\boldsymbol{\theta}_y^2] = \frac{1}{B}\left[\frac{1}{p_1} + \frac{1}{kp_2} - C\right] \tag{3.124}$$

where $p_1 = p_d(b|a)+p_{dc}(b|x)+p_d(y|x)+p_{dc}(y|a)$ and $p_2 = p_n(b|a)+p_{nc}(b|x)+p_n(y|x)+p_{nc}(y|a)$.

Proof. The optimal solution $\boldsymbol{\theta}^B$ implies the gradient $\nabla J^B_{(a,x)}(\boldsymbol{\theta}^B) = \mathbf{0}$, which gives

$$\nabla J^B_{(a,x)}(\boldsymbol{\theta}^B) = \nabla J^B_{(a,x)}(\boldsymbol{\theta}^*) + \nabla^2 J^B_{(a,x)}(\boldsymbol{\theta}^*)(\boldsymbol{\theta}^B - \boldsymbol{\theta}^*) + O(\|\boldsymbol{\theta}^B - \boldsymbol{\theta}^*\|^2) = \mathbf{0}. \tag{3.125}$$

Thus, up to terms of order $O(\|\boldsymbol{\theta}^B - \boldsymbol{\theta}^*\|^2)$, we have

$$\sqrt{B}(\boldsymbol{\theta}^B - \boldsymbol{\theta}^*) = -\left(\nabla^2 J^B_{(a,x)}(\boldsymbol{\theta}^*)\right)^{-1}\sqrt{B}\nabla J^B_{(a,x)}(\boldsymbol{\theta}^*) \tag{3.126}$$

Next we analyze $-\left(\nabla^2 J^B_{(a,x)}(\boldsymbol{\theta}^*)\right)^{-1}$ and $\sqrt{B}\nabla J^B_{(a,x)}(\boldsymbol{\theta}^*)$.

For $\left(\nabla^2 J^B_{(a,x)}(\boldsymbol{\theta}^*)\right)^{-1}$: The gradient and Hessian of $J^B_{(a,x)}$ can be computed as

$$\begin{aligned}\nabla J^B_{(a,x)}(\boldsymbol{\theta}) = &\frac{1}{B}\left[\sum_{i_1}(\sigma(\boldsymbol{\theta}_{b_{i_1}})-1)\mathbf{e}_{(b_{i_1})} + \sum_{i_2}(\sigma(\boldsymbol{\theta}_{b_{i_2}})-1)\mathbf{e}_{(b_{i_2})} + \sum_{i_3}\sigma(\boldsymbol{\theta}_{b_{i_3}})\mathbf{e}_{(b_{i_3})} + \sum_{i_4}\sigma(\boldsymbol{\theta}_{b_{i_4}})\mathbf{e}_{(b_{i_4})}\right] \\ &+ \frac{1}{B}\left[\sum_{j_1}(\sigma(\boldsymbol{\theta}_{y_{j_1}})-1)\mathbf{e}_{(y_{j_1})} + \sum_{j_2}(\sigma(\boldsymbol{\theta}_{y_{j_2}})-1)\mathbf{e}_{(y_{j_2})} + \sum_{j_3}\sigma(\boldsymbol{\theta}_{y_{i_3}})\mathbf{e}_{(y_{j_3})} + \sum_{j_4}\sigma(\boldsymbol{\theta}_{y_{j_4}})\mathbf{e}_{(y_{j_4})}\right] \\ \nabla^2 J^B_{(a,x)}(\boldsymbol{\theta}) = &f(b,i_1)+f(b,i_2)+f(b,i_3)+f(b,i_4)f(y,j_1)+f(y,j_2)+f(y,j_3)+f(y,j_4)\end{aligned} \tag{3.127}$$

where for example $f(b, i_1) = \frac{1}{B}\sum_{i_1}\sigma(\boldsymbol{\theta}_{b_{i_1}})\left(1-\sigma(\boldsymbol{\theta}_{b_{i_1}})\right)\mathbf{e}_{(b_{i_1})}\mathbf{e}'_{(b_{i_1})}$ and $e_{(b_{i_1})}$ is a one-hot vector which has only a 1 on the corresponding dimension. According to Lemma 3.8, by denoting $\mathbf{H}_{(a,x)} = \lim_{B\to+\infty}\triangledown^2 J^B_{(a,x)}(\boldsymbol{\theta}^*)$ we have at $\boldsymbol{\theta} = \boldsymbol{\theta}^*$

$$
\begin{aligned}
\mathbf{H}_{(a,x)} \xrightarrow{P} & \sum_b \sigma(\boldsymbol{\theta}^*_b)(1-\sigma(\boldsymbol{\theta}^*_b))\mathbf{e}_{(b)}\mathbf{e}'_{(b)}[p_d(b|a)+p_{dc}(b|x)+p_n(b|a)+p_{nc}(b|x)] \\
& + \sum_y \sigma(\boldsymbol{\theta}^*_y)(1-\sigma(\boldsymbol{\theta}^*_y))\mathbf{e}_{(y)}\mathbf{e}'_{(y)}[p_d(y|x)+p_{dc}(y|a)+p_n(y|x)+p_{nc}(y|a)] \\
& = \sum_{b\in\bar{\mathcal{L}}_1}\frac{k[p_d(b|a)+p_{dc}(b|x)][p_n(b|a)+p_{nc}(b|x)]}{p_d(b|a)+p_{dc}(b|x)+p_n(b|a)+p_{nc}(b|x)}\mathbf{e}_{(b)}\mathbf{e}'_{(b)} + \sum_{b\in\mathcal{L}_1}\frac{kp_1p_2}{p_1+p_2}\mathbf{e}_{(b)}\mathbf{e}'_{(b)} \\
& + \sum_{y\in\bar{\mathcal{L}}_2}\frac{k[p_d(y|x)+p_{dc}(y|a)][p_n(y|x)+p_{nc}(y|a)]}{p_d(y|x)+p_{dc}(y|a)+p_n(y|x)+p_{nc}(y|a)}\mathbf{e}_{(y)}\mathbf{e}'_{(y)} + \sum_{y\in\mathcal{L}_2}\frac{kp_1p_2}{p_1+p_2}\mathbf{e}_{(y)}\mathbf{e}'_{(y)} \\
& = \mathrm{diag}(\mathbf{m})
\end{aligned}
\tag{3.128}
$$

where $p_1 = p_d(b|a)+p_{dc}(b|x)+p_d(y|x)+p_{dc}(y|a)$ and $p_2 = p_n(b|a)+p_{nc}(b|x)+p_n(y|x)+p_{nc}(y|a)$.

We analyze $\triangledown J^B_{(a,x)}(\boldsymbol{\theta}^*)$ expectation and variance as follows.

$$
\begin{aligned}
\mathbb{E}[\triangledown J^B_{(a,x)}(\boldsymbol{\theta}^*)] = & \sum_{b\in\bar{\mathcal{L}}_1}\left[p_d(b|a)+p_{dc}(b|x)](\sigma(\boldsymbol{\theta}^*_b)-1)\mathbf{e}_{(b)} + k[p_n(b|a)+p_{nc}(b|x)]\sigma(\boldsymbol{\theta}^*_b)\mathbf{e}_{(b)}\right] \\
& + \sum_{y\in\bar{\mathcal{L}}_2}\left[p_d(y|x)+p_{dc}(y|a)](\sigma(\boldsymbol{\theta}^*_y)-1)\mathbf{e}_{(y)} + k[p_n(y|x)+p_{nc}(y|a)]\sigma(\boldsymbol{\theta}^*_y)\mathbf{e}_{(y)}\right] \\
& + \sum_{b\in\mathcal{L}_1} p_1(\sigma(\boldsymbol{\theta}^*_b)-1)\mathbf{e}_{(b)} + kp_2\sigma(\boldsymbol{\theta}_b)\mathbf{e}_{(b)} \\
& + \sum_{y\in\mathcal{L}_2} p_1(\sigma(\boldsymbol{\theta}^*_y)-1)\mathbf{e}_{(y)} + kp_2\sigma(\boldsymbol{\theta}_y)\mathbf{e}_{(y)} \\
& = \mathbf{0}
\end{aligned}
\tag{3.129}
$$

$$
\mathrm{Cov}[\triangledown J^B_{(a,x)}(\boldsymbol{\theta}^*)] = \mathbb{E}\left[\triangledown J^B_{(a,x)}(\boldsymbol{\theta}^*)(\triangledown J^B_{(a,x)}(\boldsymbol{\theta}^*))'\right] = \frac{1}{B}\left(\mathrm{diag}(\mathbf{m}) - (1+\frac{1}{k})\mathbf{m}\mathbf{m}'\right) \tag{3.130}
$$

Then, with $\mathbf{H}_{(a,x)}$ and $\mathrm{Cov}[\triangledown J^B_{(a,x)}(\boldsymbol{\theta}^*)]$, we can derive the covariance of $\sqrt{B}(\boldsymbol{\theta}^B - \boldsymbol{\theta}^*)$ as

$$
\begin{aligned}
\mathrm{Cov}\left[\sqrt{B}(\boldsymbol{\theta}^B-\boldsymbol{\theta}^*)\right] &= \mathbb{E}\left[\sqrt{B}(\boldsymbol{\theta}^B-\boldsymbol{\theta}^*)\sqrt{B}(\boldsymbol{\theta}^B-\boldsymbol{\theta}^*)'\right] \\
&\approx B\mathrm{diag}(\mathbf{m})^{-1}\mathrm{Var}\left[\triangledown J^B_{(a,x)}(\boldsymbol{\theta}^*)\right](\mathrm{diag}(\mathbf{m})^{-1})' \\
&= \mathrm{diag}(\mathbf{m})^{-1} - (1+\frac{1}{k})\mathbf{1}\mathbf{1}'
\end{aligned}
\tag{3.131}
$$

This implies that the mean square errors for non-anchor nodes $b \in \bar{\mathcal{L}}_1$ and $y \in \bar{\mathcal{L}}_2$ can be

computed by

$$\begin{aligned}\mathbb{E}\left[\Delta\boldsymbol{\theta}_b^2\right] &= \frac{1}{B}\left[\frac{1}{p_d(b|a)+p_{dc}(b|x)}+\frac{1}{kp_n(b|a)+kp_{nc}(b|x)}-C\right]\\ \mathbb{E}[\Delta\boldsymbol{\theta}_y^2] &= \frac{1}{B}\left[\frac{1}{p_d(y|x)+p_{dc}(y|a)}+\frac{1}{kp_n(y|x)+kp_{nc}(y|a)}-C\right]\end{aligned} \tag{3.132}$$

For anchor nodes $b \in \mathcal{L}_1$ and $y \in \mathcal{L}_2$, the mean square error is computed by

$$\mathbb{E}\left[\Delta\boldsymbol{\theta}_b^2\right] = \mathbb{E}[\Delta\boldsymbol{\theta}_y^2] = \frac{1}{B}\left[\frac{1}{p_1}+\frac{1}{kp_2}-C\right] \tag{3.133}$$

where $\Delta\boldsymbol{\theta}_b = \boldsymbol{\theta}_b^B - \boldsymbol{\theta}_b^*, \Delta\boldsymbol{\theta}_y = \boldsymbol{\theta}_y^B - \boldsymbol{\theta}_y^*$. This completes the proof. QED.

Given the above Lemma 3.9, the question now comes to *how to design these distributions*. For a single network where only p_d and p_n are considered, [156] proposes that p_n is positively correlated to p_d. And if node-b has a high embedding similarity with node-a, it is likely to be a negative sample. This can be considered as a *hard* negative sample which in the task of recommendation could separate the negative items from positive ones for a certain user. However, in pairwise network alignment where we have two input networks, different sampling distributions serve different purposes, which as we show in the following may lead to the competing designs. First (for p_d), a typical goal of p_d is to sample nodes that are similar to the center nodes such that the sampled nodes are likely to co-occur with the center nodes in some manually extracted contexts [160, 161]. Second (for p_n), we follow the intuition in unsupervised network embedding that close neighbors should be similar while distant nodes should be dissimilar in terms of embedding vectors [158]. This implies that distant/dissimilar nodes are more likely to be sampled by p_n. Third (for p_{dc}), we use it to sample positive alignment pairs across networks that are likely to form the alignments and preserve the *alignment consistency* similar to Eq. (3.91). This implies p_{dc} should be positively correlated to the embedding similarities. Fourth (for p_{nc}), we first note that network alignment can be considered as a special recommendation task where the anchor link of two nodes is analogized as the *only* positive item for a user. In this way, we would like to use p_{nc} to provide *hard* negative alignment pairs as in recommendation [156, 162]. That is, nodes in $\mathcal{G}_1$ that currently have high embedding similarities with anchor node-x are likely to be the hard negative samples. By doing so, we could attain the *alignment disparity*. That is, the embeddings of the nodes that are likely to mislead the alignments are encouraged to be more separable from node-x. We remark that while selecting nodes as the negative alignment pairs, they are supposed not to violate the overall alignment consistency.

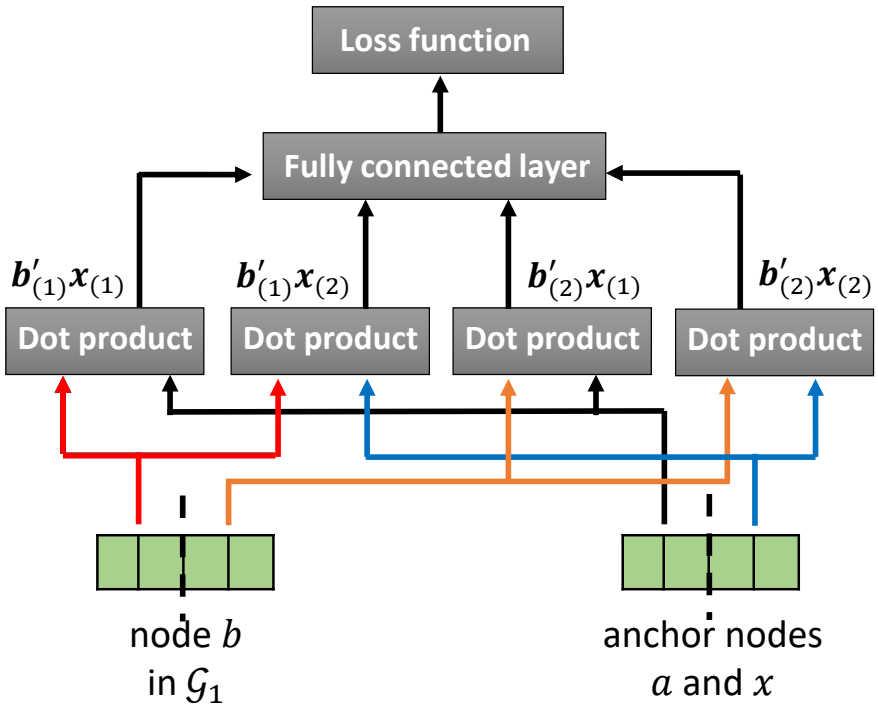


Figure 3.26: Illustration of embedding interactions.

In addition, given Lemma 3.9 and the design principles of p_d, p_n, p_{dc}, in order to estimate high-probability node pairs (i.e., high p_d, p_{dc} and low p_n) more accurately, we need p_{nc} to be large which coincides with the designed positive correlation.

But with these designs, nodes that have high embedding similarities to anchor nodes are likely to be sampled as both positive context/alignment pairs and negative alignment pairs. In other words, suppose node-b is sampled to form a positive context pair with anchor node-a and a negative alignment pair with the anchor node-x simultaneously. Then according to Eq. (3.112), it means that the node pair (b, x) should be classified into both the positive class and the negative class, i.e., two competing objectives. To address this issue, instead of simply using $\mathbf{b}'\mathbf{x}$ to compute p_d, p_n, p_{dc}, p_{nc}, we divide the node embedding vectors to two parts, i.e., $\mathbf{b} = [\mathbf{b}_{(1)} \| \mathbf{b}_{(2)}]$ and $\mathbf{x} = [\mathbf{x}_{(1)} \| \mathbf{x}_{(2)}]$ where $\mathbf{b}_{(1)}, \mathbf{b}_{(2)}, \mathbf{x}_{(1)}, \mathbf{x}_{(2)} \in \mathbb{R}^{d/2}$. Each part of the vector aims to capture different information. For example, $\mathbf{b}_{(1)}$ captures the local neighborhood information of node-b in $\mathcal{G}_1$ while $\mathbf{b}_{(2)}$ encodes how node-b posits in the context of $\mathcal{G}_2$. Then we can design a new alignment scoring function that allows the interactions among different parts of embeddings as shown in Figure 3.26. Specifically, we define

$$\mathbf{b} \star \mathbf{x} = w_1 \mathbf{b}'_{(1)} \mathbf{x}_{(1)} + w_2 \mathbf{b}'_{(1)} \mathbf{x}_{(2)} + w_3 \mathbf{b}'_{(2)} \mathbf{x}_{(1)} + w_4 \mathbf{b}'_{(2)} \mathbf{x}_{(2)} \tag{3.134}$$

where $[w_1, w_2, w_3, w_4]$ is a vector of parameters to be learned that measure the importance

of different terms. By replacing the original inner products in $J^B_{(a,x)}$ with Eq. (3.134), the embedding similarities are determined by different aspects. In particular, since $\mathbf{a} = \mathbf{x}$, the first term $\mathbf{b}'_{(1)}\mathbf{x}_{(1)} = \mathbf{b}'_{(1)}\mathbf{a}_{(1)}$ implies the intra-network proximity between node-b in $\mathcal{G}_1$ and anchor node-x and hence can be used in the sampling probabilities $p_d(b|a)$ and $p_n(b|a)$. Second, the last term $\mathbf{b}'_{(2)}\mathbf{x}_{(2)}$ describes how likely, in the context of $\mathcal{G}_2$, node-b interacts with anchor node-x, and thus can be used to measure to what extent they are aligned. In this way, this term can be used in the sampling distribution $p_{dc}(b|x)$. Lastly, the middle two terms capture how likely that two nodes b and x are interacted in a way similar as in recommendation. For example, the term $\mathbf{b}'_{(1)}\mathbf{x}_{(2)}$ can be considered as the way we do inner products in social recommendation as $\mathbf{b}_{(1)}$ and $\mathbf{x}_{(2)}$ are the embeddings of two nodes in the context of their own networks. This allows us to use this two terms to formulate the cross-network negative sampling distribution $p_{nc}(b|x)$ to provide hard negative samples. Consequently, the probability distributions for $\mathcal{G}_1$ can be formulated as

$$\begin{aligned} p_d(b|a) &= \frac{\sigma(\mathbf{b}'_{(1)}\mathbf{x}_{(1)})}{\sum_{c\in\mathcal{V}_1}\sigma(\mathbf{c}'_{(1)}\mathbf{x}_{(1)})}, \quad p_n(b|a) = \frac{\sigma(-\mathbf{b}'_{(1)}\mathbf{x}_{(1)})}{\sum_{c\in\mathcal{V}_1}\sigma(-\mathbf{c}'_{(1)}\mathbf{x}_{(1)})} \\ p_{dc}(b|x) &= \frac{\sigma(\mathbf{b}'_{(2)}\mathbf{x}_{(2)})}{\sum_{c\in\mathcal{V}_1}\sigma(\mathbf{c}'_{(2)}\mathbf{x}_{(2)})}, \quad p_{nc}(b|x) = \frac{\sigma(\mathbf{b}'_{(1)}\mathbf{x}_{(2)} + \mathbf{b}'_{(2)}\mathbf{x}_{(1)})}{\sum_{c\in\mathcal{V}_1}\sigma(\mathbf{c}'_{(1)}\mathbf{x}_{(2)} + \mathbf{c}'_{(2)}\mathbf{x}_{(1)})} \end{aligned} \tag{3.135}$$

and for $\mathcal{G}_2$ they can be computed similarly. In this way, we can encode different aspects of the sampling design principles simultaneously and strike a balance among them through the learning process. Since the real p_d is often unknown and we define its approximation by node2vec [161] to explicitly provide positive context pairs. In addition, as aforementioned, the true positive alignment pair for an anchor node is the anchor link itself. Thus the nodes sampled by $p_{dc}(b|x)$ can only be considered to form the *intermediate* positive alignment pairs. In this way, we add another loss term in $J^B_{(a,x)}$ to encode the differences between the anchor links and intermediate positive alignment pairs, which gives the final loss

$$\begin{aligned} J^B_{(a,x)} = &\frac{1}{B}\left[\sum_{i_1,i_2}\log\sigma(\mathbf{b}_{i_1}\star\mathbf{x}) + \log\sigma(\mathbf{b}_{i_2}\star\mathbf{x}) + \max\{0, \sigma(\mathbf{b}_{i_2}\star\mathbf{x}) - \sigma(\mathbf{x}\star\mathbf{x}) + \lambda\}\right] \\ &+ \frac{1}{B}\left[\sum_{j_1,j_2}\log\sigma(\mathbf{y}_{j_1}\star\mathbf{x}) + \log\sigma(\mathbf{y}_{j_2}\star\mathbf{x}) + \max(0, \sigma(\mathbf{y}_{j_2}\star\mathbf{x}) - \sigma(\mathbf{x}\star\mathbf{x}) + \lambda)\right] \\ &+ \frac{1}{B}\left[\sum_{i_3,i_4}\log\sigma(-\mathbf{b}_{i_3}\star\mathbf{x}) + \log\sigma(-\mathbf{b}_{i_4}\star\mathbf{x}) + \sum_{j_3,j_4}\log\sigma(-\mathbf{y}_{j_3}\star\mathbf{x}) + \log\sigma(-\mathbf{y}_{j_4}\star\mathbf{x})\right]. \end{aligned} \tag{3.136}$$

Table 3.10: Data statistics.

Scenarios	Networks	# of nodes	# of edges	# of attributes
S1	ACM	9,872	39,561	17
	DBLP	9,916	44,808	17
S2	Foursquare	5,313	54,233	0
	Twitter	5,120	130,575	0
S3	Phone	1,000	41,191	0
	Email	1,003	4,627	0

3.4.3 Experimental Evaluations

We evaluate the proposed NeXtAlign in the following aspects:

- Q1. How accurate is NeXtAlign for the task of network alignment?
- Q2. To what extent does NeXtAlign benefit from different components of the model?

Experimental Setup. We introduce the experimental setups as follows[7].
Datasets. The statistics of the datasets are summarized in Table 3.10. We evaluate the performance of network alignment in three different scenarios, and the datasets that we use to construct them are described as below.

- *S1 - ACM vs. DBLP.* In this scenario, we want to align two undirected co-author networks ACM and DBLP that are extracted from the papers in four areas (DM, ML, DB and IR) and their corresponding citation information [150]. In these co-author networks, nodes represent authors and there exists an edge between two nodes if they are co-authors of at least one paper. Specifically, the ACM co-author network has 9,872 nodes and 39,561 edges. The DBLP co-author network has 9,916 nodes and 44,808 edges. The attributes of each node indicate the number of papers that are published in different venues by that author. There exist 6,325 common authors across two networks used as the ground-truth alignments [12].
- *S2 - Foursquare-Twitter.* In this scenario, we want to align two social networks of Foursquare and Twitter. Each node represent a user and edges indicate the friendships among users. There are 5,313 nodes and 5,120 edges in the Foursquare network. And the Twitter network has 5,120 nodes and 130,575 edges. Node attributes are not available in these two networks. In addition, there are 1,609 common users which are used as the ground-truth alignments [130].

[7]The code can be found at `https://github.com/sizhang92/NextAlign-KDD21`.

Table 3.11: Results with 20% training data.

	ACM-DBLP		Foursquare-Twitter		Phone-Email	
	Hits@10	Hits@30	Hits@10	Hits@30	Hits@10	Hits@30
NeXtAlign	**0.842±0.003**	**0.901±0.008**	**0.296±0.010**	**0.417±0.007**	**0.393±0.017**	**0.675±0.011**
Bright	0.790±0.004	0.867±0.004	0.250±0.015	0.321±0.010	0.257±0.009	0.534±0.009
NetTrans	0.793±0.007	0.836±0.008	0.247±0.004	0.346±0.010	0.265±0.003	0.533±0.008
FINAL	0.677±0.008	0.824±0.010	0.236±0.009	0.346±0.009	0.220±0.015	0.459±0.018
IONE	0.748±0.013	0.845±0.010	0.162±0.011	0.292±0.021	0.378±0.013	0.644±0.008
CrossMNA	0.653±0.004	0.790±0.004	0.024±0.017	0.075±0.038	0.154±0.004	0.405±0.012

- *S3 - Phone-Email.* In this scenario, we aim to align the communication networks through different channels. In particular, the Phone network corresponds to the communications among people via phone, while the Email network describes the communications by emails. More specifically, there exist 1,000 nodes and 41,191 edges in the Phone network while the Email network is sparser with 1,003 nodes and 4,627 edges. In addition, there are 1,000 common people that are involved in both communication networks used as the ground-truth alignments [16].

Besides, in *S1-S3*, we evaluate with different training ratios (i.e., 10% and 20%). For example, with the training ratio as 10%, we randomly select 10% of the ground-truth alignments as the training data (i.e., anchor links) and test on the rest of the ground-truth alignments. We randomly generate 10 sets of training data for each alignment scenario. We evaluate the performance of all methods, and report the mean values and standard deviations.

Baseline methods. We compare the proposed method NeXtAlign with the following semi-supervised network alignment methods: (1) Bright [32], (2) NetTrans [14], (3) semi-supervised FINAL [9], (4) IONE [8], and (5) CrossMNA [27].

Machine. The model is implemented in Pytorch with one Nvidia GTX 1080 as GPU.

Hyperparameters settings. We use Adam optimizer with a learning rate 0.05 to train the model. We use the same hyperparameter setting in all the three alignment scenarios. Specifically, we set $\alpha = 0.5, \lambda = 0.1$. In addition, we set the batch size as 300 and the number of negative samples as $k = 20$. We train the model in 50 epochs. For all embedding based methods, we learn node embeddings with the dimension $d = 128$. The parameters in all baseline methods are set to their defaults.

Metrics. We evaluate the effectiveness of network alignment in terms of Hits@K. Given a test pair (u, v), if node-v in $\mathcal{G}_2$ is among the top-K most similar nodes to node-u in $\mathcal{G}_1$, we view it as a *hit*. Then Hits@K is computed by $\text{Hits@}K = \frac{\text{\# of hits}}{\text{\# of testing alignments}}$.

Effectiveness Results. We evaluate the performance with and without node attributes.

Table 3.12: Results with 10% training data.

	ACM-DBLP		Foursquare-Twitter		Phone-Email	
	Hits@10	Hits@30	Hits@10	Hits@30	Hits@10	Hits@30
NeXtAlign	**0.724±0.004**	**0.816±0.002**	**0.195±0.012**	**0.297±0.015**	**0.279±0.017**	**0.545±0.019**
Bright	0.702±0.007	0.764±0.007	0.171±0.003	0.245±0.010	0.203±0.013	0.453±0.010
NetTrans	0.639±0.008	0.740±0.010	0.158±0.002	0.241±0.008	0.177±0.009	0.409±0.011
FINAL	0.460±0.010	0.649±0.007	0.139±0.009	0.237±0.012	0.175±0.012	0.386±0.016
IONE	0.477±0.018	0.611±0.017	0.069±0.014	0.167±0.022	0.180±0.009	0.444±0.009
CrossMNA	0.369±0.007	0.506±0.007	0.024±0.004	0.078±0.009	0.114±0.009	0.351±0.013

Table 3.13: Alignment on ACM-DBLP with attributes.

	10% training data		20% training data	
	Hits@10	Hits@30	Hits@10	Hits@30
NeXtAlign	**0.785±0.010**	**0.871±0.009**	**0.872±0.016**	**0.942±0.003**
Bright	0.781±0.004	0.862±0.003	0.797±0.004	0.870±0.006
NetTrans	0.708±0.004	0.846±0.009	0.841±0.010	0.916±0.013
FINAL	0.651±0.013	0.817±0.009	0.825±0.008	0.916±0.006

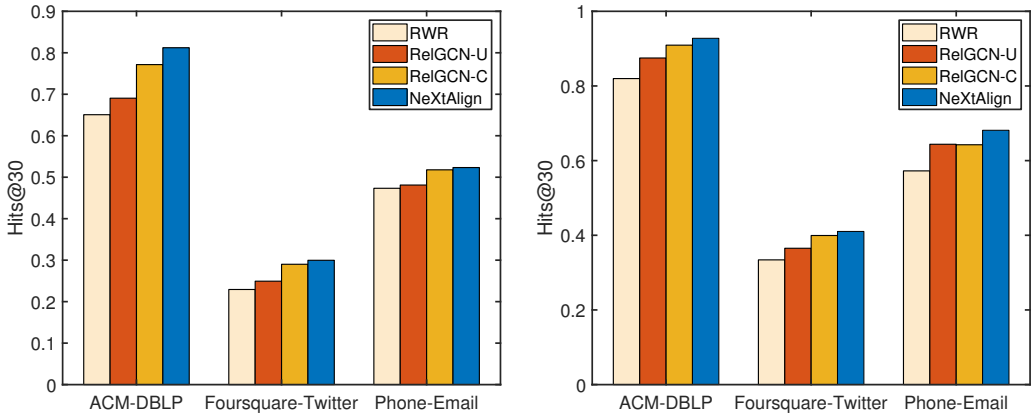
Alignment without node attributes. We first evaluate the alignment performance without using node attributes under different training ratios. The results of the experiments using 20% and 10% training data are summarized in Table 3.11 and Table 3.12 respectively. We have the following observations. First, by comparing with the consistency-based semi-supervised FINAL, our proposed method achieves an up to 20% improvement in both Hits@30 and Hits@10, which indicates that despite their close relationships in capturing the alignment consistency, our proposed method benefits from encompassing alignment disparity. Second, our proposed method consistently outperforms all the other embedding based alignment methods (i.e., Bright, NetTrans, IONE and CrossMNA). In particular, our method achieves an at least 3% improvement in Hits@30 compared to the best competitor. This demonstrates that our method can learn more meaningful node embeddings for the task of network alignment. Third, in the scenarios *S2* and *S3* where networks to be aligned are disparate with each other in terms network structure (e.g., significant differences in edge density), our proposed method achieves more improvements over the baseline methods than in the scenario *S1*. This implies that our method can perform better to align networks where alignment consistency might not be much helpful. Lastly, even with fewer training data (i.e., 10% training data), our method still outperforms other baseline methods.

Alignment with node attributes. Moreover, we evaluate the performance of node attributed network alignment in *S1*. The results are shown in Table 3.13. As we can see, all the methods benefit a lot from leveraging node attributes to infer more accurate node alignments. In the meanwhile, our method still outperforms all baseline methods under different training ratios.

Table 3.14: Ablations study on sampling strategies by Hits@30.

	ACM-DBLP	Foursquare-Twitter	Phone-Email
NeXtAlign	0.9277	0.4103	0.6813
Uniform	0.8975	0.3924	0.6525
Degree	0.9093	0.3923	0.6637
Positive	0.9097	0.4040	0.6650

Ablation Studies. We also want to validate different components of the proposed method. *Ablation study on model design.* We compare our proposed method with the following variants: (1) RWR, which uses initial embeddings with pre-positioning by random walk with restart (e.g., $\mathbf{u}^0, \mathbf{v}^0$), (2) RelGCN-U, which uses the output embedding by RelGCN-U layer as the output node embeddings, and (3) RelGCN-C, which uses the re-scaled relative positions as the final embeddings (e.g., $\tilde{\mathbf{u}}$). The results are shown in Figure 3.27. As we can see, the proposed method NeXtAlign performs the best, validating the necessities of all components in the whole model.



(a) 10% training data. (b) 20% training data.

Figure 3.27: Ablation study on model architecture.

Ablation study on sampling strategies. To demonstrate that our proposed sampling method is indeed beneficial, we compare our proposed method with different variants by changing the sampling strategies. Specifically, we compare with the model variants that for an anchor node-x, (1 - uniform) uniformly at random sample negative context pairs and alignment pairs, (2 - degree) sample negative pairs based on the node degree (e.g., $p_n(v|x) \propto d_v^{3/4}$ and $p_{nc}(u|x) \propto d_u^{3/4}$), and (3 - positive) sample nodes by the distribution which is positively correlated to the inner product among node embeddings (e.g., $p_n(v|x) \propto \sigma(\mathbf{v}'\mathbf{x})$ and $p_{nc}(u|x) \propto \sigma(\mathbf{u}'\mathbf{x})$). The results are summarized in Table 3.14. We observe that the

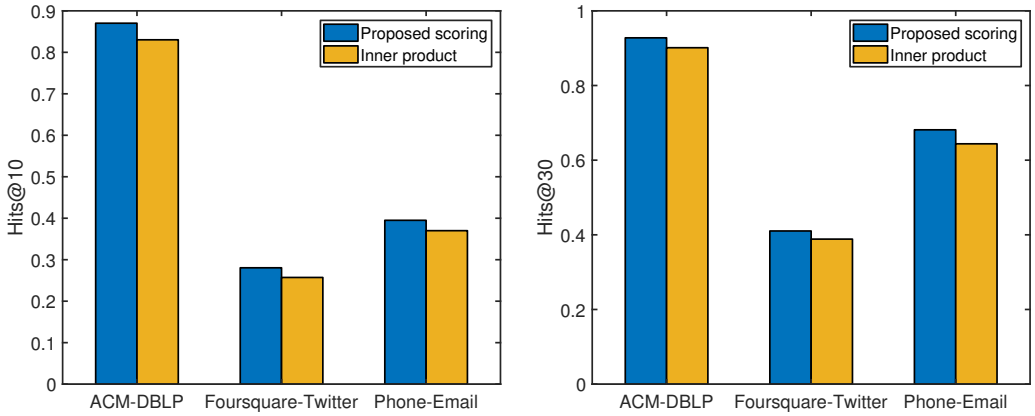
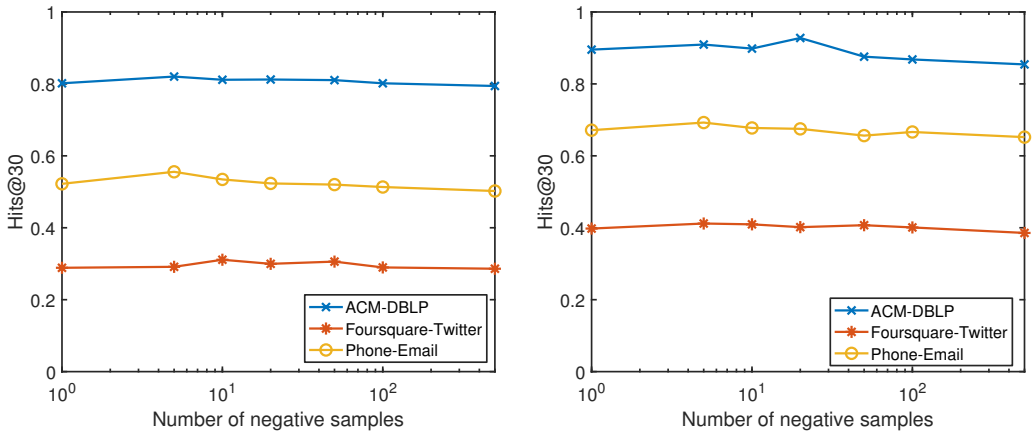


Figure 3.28: Ablation study on the scoring function.



(a) 10% training data. (b) 20% training data.

Figure 3.29: Hits@30 with different sizes of negative samples.

pre-defined sampling strategies perform the worst. While using positively correlation helps improve the alignment performance against the pre-defined distributions, our proposed sampling strategy still performs the best. This validates the benefits of using both negative correlation and positive correlation.

Ablation study on Eq. (3.134). Here, we compare the alignment performance with the model variant by replacing the scoring function Eq. (3.134) with inner product using 20% training data. As we can see in Figure 3.28, using Eq. (3.134) indeed boosts the performance.

Parameter study on the number of negative samples. We analyze how alignment performance varies with different number of negative samples $k = [1, 5, 10, 20, 50, 100, 500]$ on different datasets. The comparisons are shown in Figure 3.29. As we can see, the alignment performance is stable under different settings of sampling size k. In addition, using a relative small size of negative samples (i.e., $k \in [5, 20]$) achieves a good overall performance.

CHAPTER 4: VERACITY IN BIG NETWORK ALIGNMENT

Many real-world networks are intrinsically incomplete with missing edges due to the difficulties in data collections, and often have noise on the network structures. In this chapter, we present our works to handle the challenges related to the veracity characteristic, including (1) incomplete network alignment that jointly solves network alignment and network completion tasks [10, 16] and (2) the robustness analysis of our proposed attributed network alignment algorithm [9] against the structural noise [12].

4.1 INCOMPLETE NETWORK ALIGNMENT

Networks are prevalent and naturally appear in many areas. More often than not, in the big data era, networks in many high-impact applications are collected from *multiple* sources (i.e., *variety*), such as social networks from different social platforms, protein-protein interaction (PPI) networks from multiple tissues, transaction networks from multiple financial institutes, etc. In order to integrate the considerable information associated with multiple networks, network alignment is of key importance to find the node correspondence across networks. For example, by aligning the same users in different transaction networks, the transaction patterns of users can be comprehended to enhance the financial fraud detection. However, real-world networks are often *incomplete* (i.e., *veracity*) due to, for instance, the difficulties in data collections. As such, network completion (e.g., to infer the missing links) has become another key task which benefits many graph mining applications by providing higher-quality networks if handled properly.

Although the multi-sourced and incomplete characteristics often *co-exist* in many real networks, the state-of-the-arts have been largely addressing network alignment and network completion problems *in parallel*. For example, most of the prior network alignment methods based on topological consistency have implicitly assumed that the topology of the input networks for alignment are perfectly known a priori [6, 18]. On the other hand, the prior network completion methods aim to infer the missing links in either a single network (e.g., by matrix completion [37]) or multiple networks that are aligned beforehand (e.g., by tensor completion [163]). How can we align two input incomplete networks with missing edges?

A natural choice could be *completion-then-alignment*. That is, we first separately complete the missing edges in the input networks by some prior network completion methods, followed by the alignment across the resulting complete networks. However, there exist some fundamental limits of this strategy on the alignment performance. First (*alignment*

accuracy), the promise of this strategy lies in that by inferring the missing links of each input network, it would provide higher-quality input networks for the alignment task. However, the completion task itself might introduce noise (e.g., truly nonexistent edges), which might compromise, or even prevail the benefits of the correctly inferred missing links for the alignment task. Second (*alignment efficiency*), the network alignment alone is already computationally costly. Most of the prior methods (even with approximation, such as [9]) have a time/space complexity at least $O(n^2)$, where n is the number of nodes of the input networks, mainly due to the computation/storage of the alignment matrix and the sparse matrix-matrix multiplication between the input adjacency matrices and the alignment matrix[1]. Yet, network completion would make each input network even denser by adding the missing edges. As a result, if we simply conduct the network alignment task on such densified networks, it might make the computation even more intensive.

To address these limitations, we hypothesize that network alignment and network completion can inherently complement each other due to the following reasons. First, *(H1) alignment helps completion.* Intuitively, when many nodes in one network share similar connectivity patterns with their corresponding aligned nodes (e.g., connecting to the similar sets of nodes) in another network, the knowledge about the existence or absence of links in one network could help inferring the missing links in another network via alignment if we can find such node correspondences across networks. Second, *(H2) completion helps alignment.* As introduced before, network completion could potentially improve the qualities of input networks, leading to the enhancement of the alignment accuracy. Moreover, network completion itself implicitly assumes a low-rank structure on the input networks, which, if harnessed appropriately, will actually *accelerate* the alignment process.

Armed with these hypotheses, we *jointly* address network alignment and network completion problems so that the two tasks could mutually benefit from each other. To be specific, in order to leverage alignment for the completion task, we impose the low-rank structure on the underlying (true) network, which matches not only the observed links of the corresponding network, but also the *auxiliary observations* from the other network via the alignment matrix. Second, in order to leverage the network completion for the alignment, we recast the network alignment problem via the low-rank structures of the *complete* networks, which not only improves the alignment accuracy, but also speeds up the alignment process. We formulate them into a joint optimization problem and develop an effective algorithm. The main contributions of this work are summarized as:

- *Problem Definition.* To our best knowledge, we are the first to jointly address the

[1]Although the *empirical* run-time of some prior methods (e.g., BigAlign [18]) is *near-linear*, the big-O time complexity of these methods is still quadratic.

network alignment and network completion tasks in an optimization framework.

- *Algorithm and Analysis.* We develop an effective algorithm (iNeat) based on the multiplicative update to solve the optimization. We also analyze its correctness, convergence and complexity. We prove that the low-rank structure of the complete networks guarantees a low-rank structure of the alignment matrix, which in turn reduces the time complexity of each iterative update to be *linear*. To our best knowledge, this is the first known network alignment algorithm with a provable linear time complexity.

- *Experiments.* We evaluate the effectiveness and efficiency of the proposed algorithm by extensive experiments. The experimental results demonstrate that (1) network alignment and network completion can indeed benefit from each other in terms of alignment accuracy and missing edges recovery rate, (2) our algorithm iNeat achieves a better alignment and completion quality, and meanwhile is faster than most of the baseline methods, and (3) our algorithm is only *linear* w.r.t. the number of nodes.

4.1.1 Problem Definition

Table 4.1 summarizes the main symbols and notations used throughout this work. We use bold uppercase letters for matrices (e.g., $\mathbf{A}$), bold lowercase letters for vectors (e.g., $\mathbf{s}$), and lowercase letters (e.g., α) for scalars. We use $\mathbf{A}(i, j)$ to denote the entry at the intersection of the i-th row and j-th column of the matrix $\mathbf{A}$. We denote the transpose of a matrix by a superscript prime (e.g., $\mathbf{A}'$ is the transpose of $\mathbf{A}$). The vectorization of a matrix is denoted by $\text{vec}(\cdot)$, and the result vector is denoted by the corresponding bold lowercase letter (e.g., $\mathbf{s} = \text{vec}(\mathbf{S})$). Equivalently, the transformation of a vector to its corresponding matrix is denoted by a de-vectorization operator $\text{mat}(\cdot)$ (e.g., $\mathbf{S} = \text{mat}(\mathbf{s})$). The trace of a matrix is denoted by $\text{Tr}(\cdot)$, and the diagonal matrix of a vector is denoted by $\text{diag}(\cdot)$.

Many real-world networks are incomplete with missing edges. Although some incompleteness scenarios may be possible (e.g., with the probabilities whether edges exist known a priori), in this work, we only consider the network incompleteness where we only have the knowledge about the existence (i.e., a value of 1) or the absence (i.e., a value of 0) of certain entries (denoted by the set Ω) of its adjacency matrix. For the rest entries in the adjacency matrix, we do not know if the corresponding links exist or not, and hence are represented as the question mark ?. Figure 4.1 presents an illustrative example. All solid lines represent the observed existing edges. As we can see in Figure 4.1 (a), the set of nodes $(1, 2, 3, 4)$ in the first incomplete network have similar topology to the nodes $(6', 7', 8', 9')$, possibly leading to a wrong alignment result that these two sets of nodes are aligned within each other. However,

Table 4.1: Symbols and notations.

Symbols	Definition
$\mathcal{G}_1$, $\mathcal{G}_2$	incomplete networks
$\mathbf{A}_1, \mathbf{A}_2$	two adjacency matrices of $\mathcal{G}_1$ and $\mathcal{G}_2$
n_1, n_2	# of nodes in $\mathcal{G}_1$ and $\mathcal{G}_2$
m_1, m_2	# of nodes in $\mathcal{G}_1$ and $\mathcal{G}_2$
$\mathbf{S}$	an $n_1 \times n_2$ alignment matrix between $\mathcal{G}_1$ and $\mathcal{G}_2$
$P_\Omega(\cdot), P_{\bar{\Omega}}(\cdot)$	an operator to project only to observed (unobserved) entries
$\mathbf{U}_1, \mathbf{V}_1, \mathbf{U}_2, \mathbf{V}_2$	low rank factorizations of $\mathbf{A}_1$ and $\mathbf{A}_2$
$\mathbf{P}_{\Omega_1}, \mathbf{P}_{\Omega_2}$	projection matrix, all 1s at all observed entries
$\mathbf{1}_1, \mathbf{1}_2$	1s vectors of length n_1 and n_2 respectively
λ, γ, β	parameters
Tr[$\cdot$]	trace operator
diag($\cdot$)	diagonal matrix of a vector
vec($\cdot$), mat($\cdot$)	vectorization and de-vectorization operator
rank($\cdot$)	the rank of a matrix
eig($\cdot$)	eigenvalues of a matrix

the complete networks in Figure 4.1 (b) (by filling all the red lines) are identical, such as the cliques formed by nodes $(1, 2, 3, 4)$ and $(1', 2', 3', 4')$. Thus, the set of nodes $(1, 2, 3, 4)$ can be aligned to nodes $(1', 2', 3', 4')$ respectively, so can the rest of nodes. On the other hand, by completing two networks separately, noisy edges might be incorrectly added (e.g., edge $(4, 6)$) and the true network structure would fail to be recovered. The incorrectly recovered networks may further mislead the alignment results. Therefore, how to align the incomplete networks while completing them is the key challenge this work aims to address.

Problem 4.1. Incomplete Network Alignment.

Given: (1) incomplete adjacency matrices $\mathbf{A}_1$, $\mathbf{A}_2$ of two undirected networks $\mathcal{G}_1, \mathcal{G}_2$, and (2-optional) a prior node similarity matrix $\mathbf{L}$ across networks.

Output: (1) the $n_1 \times n_2$ alignment/similarity matrix $\mathbf{S}$, where $\mathbf{S}(a, x)$ represents to what extent node-a in $\mathcal{G}_1$ is aligned with node-x in $\mathcal{G}_2$, and (2) complete adjacency matrices $\mathbf{A}_1^*, \mathbf{A}_2^*$.

Preliminaries. In this part, we introduce the necessary preliminaries for this study.

A - Network Alignment. Most prior network alignment algorithms, explicitly or implicitly, are based on the *topology consistency* principle. Take FINAL as an example, the topology consistency principle can be stated as follows[2]. Given two pairs of nodes, say (1) node-a in $\mathcal{G}_1$ and node-x in $\mathcal{G}_2$ and (2) node-b in $\mathcal{G}_1$ and node-y in $\mathcal{G}_2$, if nodes a and b are close neighbors

[2]In [9], we generalize the topology consistency principle to further accommodate the additional node/edge attribute information, which is outside the scope of this work.

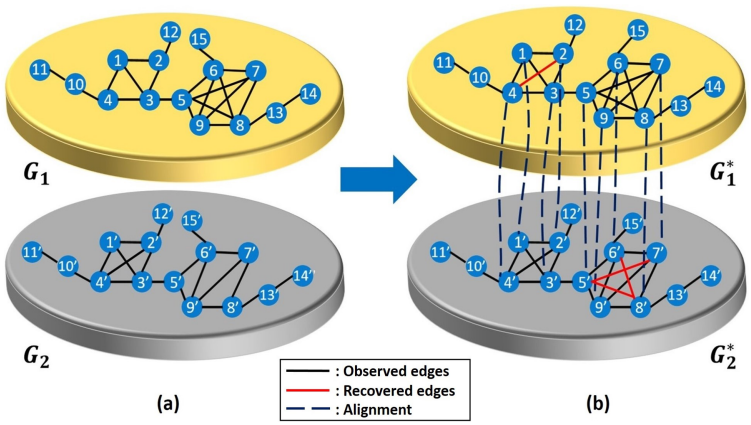


Figure 4.1: An illustrative example. (a) shows the input incomplete networks and (b) shows part of the alignment across two complete networks.

and nodes x and y are also close neighbors, the *topology consistency* principle assumes the similarity between a and x, and that between their respective close neighbors b and y to be consistent, i.e., small $[\hat{\mathbf{S}}(a,x) - \hat{\mathbf{S}}(b,y)]^2 \mathbf{A}_1(a,b)\mathbf{A}_2(x,y)$, where $\hat{\mathbf{S}}$ is the similarity matrix. Mathematically, this naturally leads to the following optimization problem:

$$\min_{\hat{\mathbf{s}}} \quad \alpha \hat{\mathbf{s}}'(\mathbf{D} - \mathbf{A}_2 \otimes \mathbf{A}_1)\hat{\mathbf{s}} + (1-\alpha)\|\mathbf{D}^{\frac{1}{2}}(\hat{\mathbf{s}} - \mathbf{l})\|_F^2 \tag{4.1}$$

where $\hat{\mathbf{s}}, \mathbf{l}$ are the vectorization of the similarity matrix $\hat{\mathbf{S}}$ and the prior similarity matrix $\mathbf{L}$ respectively. $\mathbf{D} = \mathbf{D}_2 \otimes \mathbf{D}_1$ and $\mathbf{D}_1, \mathbf{D}_2$ are the diagonal degree matrix of $\mathbf{A}_1, \mathbf{A}_2$ respectively. Note that instead of using $\hat{\mathbf{S}}$ to infer the alignment as in [9], we use the scaled similarity matrix $\mathbf{S}$ as the 'soft' alignment matrix throughout this work where $\mathbf{S}$ is the matrix form of $\mathbf{s} = \mathbf{D}\hat{\mathbf{s}}$ (i.e., $\mathbf{S} = \text{mat}(\mathbf{D}\hat{\mathbf{s}})$). In other words, the entries in the alignment matrix $\mathbf{S}$ measure to what extent the two corresponding nodes are aligned together. Besides, the second regularization term in Eq. (4.1) is to avoid trivial solutions, such as a zero matrix $\hat{\mathbf{S}}$.

In order to solve the network alignment problem in Eq. (4.1), we can either use an iterative algorithm with a time complexity of $O(nm)$ and a space complexity $O(n^2)$, or resort to its closed-form solution whose time complexity could be as high as $O(n^6)$ where we assume that the two networks have a comparable number of edges and nodes, i.e., $O(m) = O(m_1) = O(m_2)$ and $O(n) = O(n_1) = O(n_2)$. In [9], we approximate the closed-form solution via eigenvalue decomposition. But it is still *quadratic* in both time and space.

B - Network Completion. As mentioned earlier, incomplete networks might have many unobserved missing edges, which could significantly change the true network structure and hence mislead the topology-based network alignment. One straightforward way to address this issue is by using matrix completion. Most of the prior matrix completion methods

are centered around minimizing the nuclear norm of the matrix [164]. However, since real-world networks are usually very large, it is very costly to directly minimize the nuclear norm of the adjacency matrices. In [165], the authors show that the nuclear norm $\|\mathbf{A}_1\|_* = \min_{\mathbf{U}_1,\mathbf{V}_1} \frac{1}{2}(\|\mathbf{U}_1\|_F^2+\|\mathbf{V}_1\|_F^2)$ where $\mathbf{A}_1 = \mathbf{U}_1\mathbf{V}_1'$, which allows the factorization-based completion methods. To be specific, we minimize the following objective function

$$\begin{aligned} J_1(\mathbf{U}_1,\mathbf{V}_1,\mathbf{U}_2,\mathbf{V}_2) = &\underbrace{\frac{1}{2}\|P_{\Omega_1}(\mathbf{A}_1-\mathbf{U}_1\mathbf{V}_1')\|_F^2+\frac{\lambda}{2}(\|\mathbf{U}_1\|_F^2+\|\mathbf{V}_1\|_F^2)}_{\text{network completion on } \mathbf{A}_1} \\ &+\underbrace{\frac{1}{2}\|P_{\Omega_2}(\mathbf{A}_2-\mathbf{U}_2\mathbf{V}_2')\|_F^2+\frac{\lambda}{2}(\|\mathbf{U}_2\|_F^2+\|\mathbf{V}_2\|_F^2)}_{\text{network completion on } \mathbf{A}_2} \end{aligned} \tag{4.2}$$

where the operator P_{Ω_1} projects values to the observed set Ω_1 of $\mathbf{A}_1$, e.g., $P_{\Omega_1}((\mathbf{U}_1\mathbf{V}_1')(i,j)) = (\mathbf{U}_1\mathbf{V}_1')(i,j)$ for any $(i,j) \in \Omega_1$, otherwise 0; and operator P_{Ω_2} is defined similarly.

4.1.2 iNeat: Optimization Formulation

In this part, we first present how to formulate the alignment task in the form of two complete networks. A key contribution here is that we prove that the low-rank structure of the complete networks guarantees a low-rank structure of the alignment matrix. Then we present how to leverage the alignment matrix to infer missing edges across networks.

Network Completion Helps Network Alignment. By performing the network completion on both incomplete networks, the structure of the underlying networks could be recovered so that we can perform the alignment task across higher-quality networks. We use the factorization-based network completion (i.e., Eq (4.2)) and denote these two complete networks by $\mathbf{A}_1^* = \mathbf{U}_1\mathbf{V}_1'$ and $\mathbf{A}_2^* = \mathbf{U}_2\mathbf{V}_2'$, where $\mathbf{U}_i$ and $\mathbf{V}_i$ $(i = 1, 2)$ are the factorization matrices of rank-r. We adopt Eq. (4.1) to perform the network alignment task. Note that in general, we cannot guarantee the recovered adjacency matrices ($\mathbf{A}_1^*$ and $\mathbf{A}_2^*$) to be symmetric because $\mathbf{V}_1$ ($\mathbf{V}_2$) may not be identical to $\mathbf{U}_1$ ($\mathbf{U}_2$). This leads to a slightly different objective function from Eq. (4.1) to align directed networks. Specifically, based on the *topology consistency* in two directed networks, the optimization problem is formulated as

$$\min_{\hat{\mathbf{s}}} \ \alpha\hat{\mathbf{s}}'(\hat{\mathbf{D}}-\mathbf{A}_2^*\otimes\mathbf{A}_1^*)\hat{\mathbf{s}}+(1-\alpha)\|\hat{\mathbf{D}}^{\frac{1}{2}}(\hat{\mathbf{s}}-\mathbf{l})\|_F^2 \tag{4.3}$$

where $\hat{\mathbf{D}} = \frac{\mathbf{D}_2\otimes\mathbf{D}_1+\hat{\mathbf{D}}_2\otimes\hat{\mathbf{D}}_1}{2}$, $\mathbf{D}_1 = \text{diag}(\mathbf{U}_1\mathbf{V}_1'\mathbf{1}_1)$ and $\hat{\mathbf{D}}_1 = \text{diag}(\mathbf{1}_1'\mathbf{U}_1\mathbf{V}_1')$ are the outdegree matrix and indegree matrix of $\mathbf{A}_1^*$, respectively. $\mathbf{D}_2$ and $\hat{\mathbf{D}}_2$ are defined in a similar way.

However, directly solving the above problem requires at least $O(n^2)$ time complexity. To address this issue, we give the following lemma, which states the alignment matrix $\mathbf{S}$ intrinsically consists of a *low-rank* structure, thanks to the low rank of adjacency matrices.

Lemma 4.1. Low-Rank Structure of the Alignment Matrix S. Let $\hat{\mathbf{s}}$ be the solution of Eq. (4.3) where $\mathbf{A}_1^* = \mathbf{U}_1\mathbf{V}_1'$ and $\mathbf{A}_2^* = \mathbf{U}_2\mathbf{V}_2'$ are two complete rank-r adjacency matrices. Let the alignment matrix $\mathbf{S}$ be the scaled similarity matrix $\mathbf{S} = \text{mat}(\hat{\mathbf{D}}\hat{\mathbf{s}})$ and $\mathbf{L}$ be the prior similarity matrix, then if $\alpha < 0.5$, the alignment matrix can be expressed as $\mathbf{S} = \alpha\mathbf{U}_1\mathbf{M}\mathbf{U}_2 + (1-\alpha)\mathbf{L}$ where $\mathbf{M}$ is an $r_1 \times r_2$ matrix and $r_1,\ r_2$ are the ranks of $\mathbf{A}_1^*$ and $\mathbf{A}_2^*$.

Proof. By Woodbury matrix identity [108], the closed-form solution of $\hat{\mathbf{S}}$ can be computed by

$$\hat{\mathbf{s}} = (1-\alpha)\hat{\mathbf{D}}^{-1}\mathbf{l} + \alpha(1-\alpha)\hat{\mathbf{D}}^{-1}\mathbf{U}\mathbf{\Lambda}^{-1}\mathbf{V}'\hat{\mathbf{D}}^{-1}\mathbf{l} \tag{4.4}$$

where $\mathbf{U} = \mathbf{U}_2 \otimes \mathbf{U}_1$, $\mathbf{V} = \mathbf{V}_2 \otimes \mathbf{V}_1$, $\mathbf{\Lambda} = \mathbf{I} - \alpha\mathbf{V}'\hat{\mathbf{D}}^{-1}\mathbf{U}$.

First, we rewrite $\mathbf{\Lambda}^{-1}$ as follows. Since for any two matrices $\mathbf{X}, \mathbf{Y}$, the eigenvalues of their product satisfies $\text{eig}(\mathbf{XY}) = \text{eig}(\mathbf{YX})$ [108], we obtain

$$\begin{aligned} |\text{eig}(\alpha\mathbf{V}'\hat{\mathbf{D}}^{-1}\mathbf{U})| &\leq |\text{eig}(2\alpha\mathbf{U}\mathbf{V}'(\mathbf{D}_2 \otimes \mathbf{D}_1)^{-1})| \\ &= 2\alpha|\text{eig}((\mathbf{U}_2\mathbf{V}_2'\mathbf{D}_2^{-1}) \otimes (\mathbf{U}_1\mathbf{V}_1'\mathbf{D}_1^{-1}))| \end{aligned} \tag{4.5}$$

Here, the term $\mathbf{U}_1\mathbf{V}_1'\mathbf{D}_1^{-1}$ represents a weighted directed network whose adjacency matrix has eigenvalues within $(-1, 1)$, so as the term $\mathbf{U}_2\mathbf{V}_2'\mathbf{D}_2^{-1}$. Thus, if $\alpha < 0.5$, according to the spectrum property of Kronecker product, we have $2\alpha|\text{eig}((\mathbf{U}_2\mathbf{V}_2'\mathbf{D}_2^{-1}) \otimes (\mathbf{U}_1\mathbf{V}_1'\mathbf{D}_1^{-1}))| < 1$. Then, we can use Neumann expansion on $\mathbf{\Lambda}^{-1}$ as

$$\mathbf{\Lambda}^{-1} = \sum_{k=0}^{\infty}(2\alpha)^k[\mathbf{V}'(2\hat{\mathbf{D}})^{-1}\mathbf{U}]^k \tag{4.6}$$

Next, we rewrite $(2\hat{\mathbf{D}})^{-1}$ as follows. Denote $\bar{\mathbf{D}}_1 = \mathbf{D}_1 + \hat{\mathbf{D}}_1$ and $\bar{\mathbf{D}}_2 = \mathbf{D}_2 + \hat{\mathbf{D}}_2$, we have

$$\begin{aligned} (2\hat{\mathbf{D}})^{-1} &= (\mathbf{D}_2 \otimes \mathbf{D}_1 + \hat{\mathbf{D}}_2 \otimes \hat{\mathbf{D}}_1)^{-1} \\ &= [(\bar{\mathbf{D}}_2 \otimes \bar{\mathbf{D}}_1)[\mathbf{I} - (\bar{\mathbf{D}}_2^{-1} \otimes \bar{\mathbf{D}}_1^{-1})(\mathbf{D}_2 \otimes \hat{\mathbf{D}}_1 + \hat{\mathbf{D}}_2 \otimes \mathbf{D}_1)]]^{-1} \\ &= [\mathbf{I} - (\bar{\mathbf{D}}_2^{-1} \otimes \bar{\mathbf{D}}_1^{-1})(\mathbf{D}_2 \otimes \hat{\mathbf{D}}_1 + \hat{\mathbf{D}}_2 \otimes \mathbf{D}_1)]^{-1}(\bar{\mathbf{D}}_2^{-1} \otimes \bar{\mathbf{D}}_1^{-1}) \\ &= \sum_{j=0}^{\infty}[(\bar{\mathbf{D}}_2^{-1}\mathbf{D}_2) \otimes (\bar{\mathbf{D}}_1^{-1}\hat{\mathbf{D}}_1) + (\bar{\mathbf{D}}_2^{-1}\hat{\mathbf{D}}_2) \otimes (\bar{\mathbf{D}}_1^{-1}\mathbf{D}_1)]^j(\bar{\mathbf{D}}_2^{-1} \otimes \bar{\mathbf{D}}_1^{-1}) \\ &= \sum_{j=0}^{\infty}\sum_{i=0}^{j}\binom{j}{i}[(\bar{\mathbf{D}}_2^{-1}\mathbf{D}_2)^i(\bar{\mathbf{D}}_2^{-1}\hat{\mathbf{D}}_2)^{j-i}\bar{\mathbf{D}}_2^{-1}] \otimes [(\bar{\mathbf{D}}_1^{-1}\hat{\mathbf{D}}_1)^i(\bar{\mathbf{D}}_1^{-1}\mathbf{D}_1)^{j-i}\bar{\mathbf{D}}_1^{-1}] \end{aligned} \tag{4.7}$$

By substituting the above equation into Eq. (4.6), the matrix $\mathbf{\Lambda}^{-1}$ can be derived as

$$\begin{aligned}\mathbf{\Lambda}^{-1} = &\sum_{k=0}^{\infty}\sum_{j=0}^{\infty}\sum_{i=0}^{j}(2\alpha)^k\binom{j}{i}^k[\mathbf{V}_2'(\bar{\mathbf{D}}_2^{-1}\mathbf{D}_2)^i(\bar{\mathbf{D}}_2^{-1}\hat{\mathbf{D}}_2)^{j-i}\bar{\mathbf{D}}_2^{-1}\mathbf{U}_2]^k \\ &\otimes[\mathbf{V}_1'(\bar{\mathbf{D}}_1^{-1}\hat{\mathbf{D}}_1)^i(\bar{\mathbf{D}}_1^{-1}\mathbf{D}_1)^{j-i}\bar{\mathbf{D}}_1^{-1}\mathbf{U}_1]^k\end{aligned} \tag{4.8}$$

Denote $\mathbf{s} = \hat{\mathbf{D}}\hat{\mathbf{s}}$ and $\hat{\mathbf{l}} = \hat{\mathbf{D}}^{-1}\mathbf{l}$. Armed with the Kronecker product property $\text{vec}(\mathbf{ABC}) = (\mathbf{C}' \otimes \mathbf{A})\text{vec}(\mathbf{B})$, by substituting Eq. (4.8) into Eq. (4.4), we obtain the alignment matrix $\mathbf{S} = \text{mat}(\hat{\mathbf{D}}\hat{\mathbf{s}})$ as

$$\mathbf{S} = \alpha\mathbf{U}_1\mathbf{M}\mathbf{U}_2' + (1-\alpha)\mathbf{L} \tag{4.9}$$

where $\mathbf{M}$ is an $r_1 \times r_2$ matrix and is computed by

$$\begin{aligned}\mathbf{M} = (1-\alpha)&\sum_{k=0}^{\infty}\sum_{j=0}^{\infty}\sum_{i=0}^{j}2^k\alpha^k\binom{j}{i}^k[\mathbf{V}_1'(\bar{\mathbf{D}}_1^{-1}\hat{\mathbf{D}}_1)^i(\bar{\mathbf{D}}_1^{-1}\mathbf{D}_1)^{j-i}\bar{\mathbf{D}}_1^{-1}\mathbf{U}_1]^k\mathbf{V}_1'\hat{\mathbf{L}}\mathbf{V}_2 \\ &\times[\mathbf{U}_2'(\bar{\mathbf{D}}_2^{-1}\mathbf{D}_2)^i(\bar{\mathbf{D}}_2^{-1}\hat{\mathbf{D}}_2)^{j-i}\bar{\mathbf{D}}_2^{-1}\mathbf{V}_2]^k\end{aligned} \tag{4.10}$$

This completes the proof. QED.

Remarks. Eq. (4.9) suggests that the alignment matrix $\mathbf{S}$ consists of two parts, including a low-rank structure and an additive term $\mathbf{L}$ to reflect the prior knowledge and is a convex combination of these two parts. Such a convex optimization follows Eq. (4.3) where a regularization term is added to minimize the inconsistency between the alignment result and the prior information. Note that other types of regularization in Eq. (4.3) can lead to more complex combinations with the prior knowledge which may utilize both the reliable and the unreliable prior information in a better way. However, we only consider Eq. (4.3) and Eq. (4.9) in this work and leave the more complex combinations to future works.

In practice, the prior knowledge matrix $\mathbf{L}$ is either low-rank (e.g., a rank-one uniform matrix) or very sparse. Having this in mind, we will mainly focus on how to learn the true low-rank structure part of $\mathbf{S}$ (i.e., $\mathbf{U}_1\mathbf{M}\mathbf{U}_2'$) from the input incomplete networks. This naturally leads to the following effective strategy. First, we temporarily treat the low-rank structure part as the alignment matrix to be solved in the optimization problems (i.e., $\mathbf{S} \approx \mathbf{U}_1\mathbf{M}\mathbf{U}_2'$). After $\mathbf{U}_1, \mathbf{M}, \mathbf{U}_2$ are obtained, we can then calibrate the result by averaging between the learned $\mathbf{S}$ and the prior knowledge $\mathbf{L}$ to further emphasize the importance of the prior knowledge, i.e., $\mathbf{S} \leftarrow (1-\alpha)\mathbf{L} + \alpha\mathbf{S}$. As we will show in the next section, a direct benefit of this strategy is that we can reduce the overall complexity (for both space and time cost) to be *linear*.

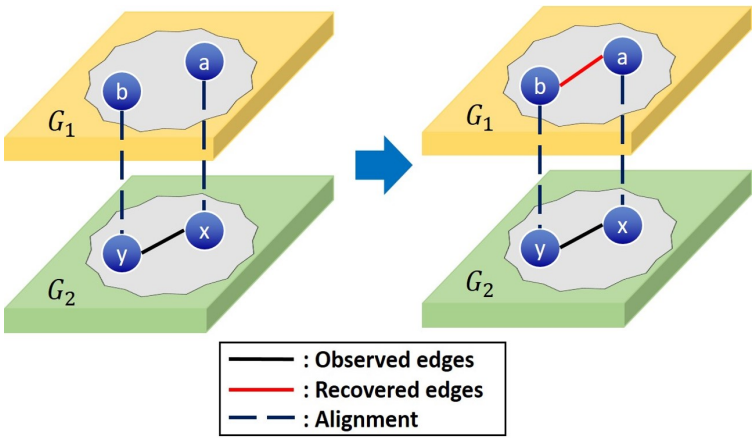


Figure 4.2: Network completion via the alignment.

To take advantages of the low-rank structure of $\mathbf{S}$ under the above strategy, instead of minimizing Eq. (4.3) regarding the similarity matrix $\hat{\mathbf{S}}$, we alternatively optimize the topology consistency on the low-rank structure of alignment matrix $\mathbf{S} = \mathbf{U}_1\mathbf{M}\mathbf{U}_2'$ without the second regularization term. Given $\mathbf{A}_1^* = \mathbf{U}_1\mathbf{V}_1'$, $\mathbf{A}_2^* = \mathbf{U}_2\mathbf{V}_2'$, by using the properties $\mathrm{vec}(\mathbf{A})'\mathrm{vec}(\mathbf{B}) = \mathrm{Tr}(\mathbf{A}'\mathbf{B})$ and $\mathrm{vec}(\mathbf{ABC}) = (\mathbf{C}' \otimes \mathbf{A})\mathrm{vec}(\mathbf{B})$, network alignment across the complete networks can be formulated as minimizing the following objective function

$$\begin{aligned} J_2(\mathbf{U}_1, \mathbf{V}_1, \mathbf{U}_2, \mathbf{V}_2, \mathbf{M}) &= \frac{\gamma}{2}\mathbf{s}'\mathrm{vec}(\mathbf{D}_1\mathbf{S}\mathbf{D}_2 + \hat{\mathbf{D}}_1\mathbf{S}\hat{\mathbf{D}}_2) - \gamma\mathbf{s}'\mathrm{vec}(\mathbf{U}_1\mathbf{V}_1'\mathbf{S}\mathbf{V}_2\mathbf{U}_2') \\ &= \underbrace{\frac{\gamma}{2}\mathrm{Tr}(\mathbf{D}_1\mathbf{U}_1\mathbf{M}\mathbf{U}_2'\mathbf{D}_2\mathbf{U}_2\mathbf{M}'\mathbf{U}_1' + \hat{\mathbf{D}}_1\mathbf{U}_1\mathbf{M}\mathbf{U}_2'\hat{\mathbf{D}}_2\mathbf{U}_2\mathbf{M}'\mathbf{U}_1')}_{\text{alignment across complete networks}} \\ &\quad - \underbrace{\gamma\mathrm{Tr}(\mathbf{U}_1\mathbf{V}_1'\mathbf{U}_1\mathbf{M}\mathbf{U}_2'\mathbf{V}_2\mathbf{U}_2'\mathbf{U}_2\mathbf{M}'\mathbf{U}_1')}_{\text{alignment across complete networks}}. \end{aligned} \tag{4.11}$$

Network Alignment Helps Network Completion. Despite the effectiveness of the factorization-based network completion methods (i.e., Eq. (4.2)), in some applications, the information of a single network alone might be insufficient to correctly infer the missing edges. Meanwhile, the alignment across the two networks may provide extra hints of how to infer the missing edges. To be specific, since the aligned nodes are likely to share similar connectivity patterns, the observed existing edges in one network could potentially help recover the missing edges in the other network via the alignment matrix. Figure 4.2 presents an illustrative example. Here, node-a in $\mathcal{G}_1$ and node-x in $\mathcal{G}_2$ are aligned together, and the neighbor of x (say node-y) is aligned with the neighbor of a (e.g., node-b), which is not observed to connect with a. If we perform the completion solely based on the observed

information of $\mathcal{G}_1$, we might probably conclude that the edge between a and b does not exist. However, the facts that (1) a and x are aligned, (2) b and y are aligned, and (3) there is an edge between x and y might provide an *auxiliary confidence* about the existence of the edge between a and b. In general, we can estimate such auxiliary confidence of the existence of the edge between a and b in $\mathcal{G}_1$ as

$$\mathbf{A}_1^*(a,b) \approx \sum_{x,y}^{n_2} \mathbf{S}(a,x)\mathbf{S}(b,y)\mathbf{A}_2(x,y) = (\mathbf{S}\mathbf{A}_2\mathbf{S}')(a,b) \tag{4.12}$$

where $\mathbf{S} = \mathbf{U}_1\mathbf{M}\mathbf{U}_2'$ is the alignment matrix learned from the topology consistency.

In the experiments, we find that such auxiliary confidence is most powerful to estimate the existence/absence of an edge (a,b) when such an edge itself is not observed in $\mathcal{G}_1$ (i.e.,$(a,b) \in \bar{\Omega}_1$). Mathematically, this can be formulated as the following objective function.

$$\begin{aligned} J_3(\mathbf{U}_1,\mathbf{V}_1,\mathbf{U}_2,\mathbf{V}_2,\mathbf{M}) = & \underbrace{\frac{\beta}{2}\|P_{\bar{\Omega}_1}(\mathbf{U}_1\mathbf{V}_1' - \mathbf{U}_1\mathbf{M}\mathbf{U}_2'\mathbf{A}_2\mathbf{U}_2\mathbf{M}'\mathbf{U}_1')\|_F^2}_{\text{completion of } \mathcal{G}_1 \text{ based on the observed edges in } \mathcal{G}_2} \\ & \underbrace{+\frac{\beta}{2}\|P_{\bar{\Omega}_2}(\mathbf{U}_2\mathbf{V}_2' - \mathbf{U}_2\mathbf{M}'\mathbf{U}_1'\mathbf{A}_1\mathbf{U}_1\mathbf{M}\mathbf{U}_2')\|_F^2}_{\text{completion of } \mathcal{G}_2 \text{ based on the observed edges in } \mathcal{G}_1} \end{aligned} \tag{4.13}$$

where $\bar{\Omega}_1$ and $\bar{\Omega}_2$ are the unobserved set of $\mathbf{A}_1$ and $\mathbf{A}_2$.

Overall Objective Function. We impose the non-negativity constraints on all the variables $\mathbf{U}_1,\mathbf{V}_1,\mathbf{U}_2,\mathbf{V}_2,\mathbf{M}$ to guarantee that all the entries in matrices $\mathbf{A}_1^*,\mathbf{A}_2^*,\mathbf{S}$ to be non-negative. The overall optimization problem is formulated as

$$\begin{aligned} \min_{\mathbf{U}_1,\mathbf{V}_1,\mathbf{U}_2,\mathbf{V}_2,\mathbf{M}} \quad & J(\mathbf{U}_1,\mathbf{V}_1,\mathbf{U}_2,\mathbf{V}_2,\mathbf{M}) = J_1 + J_2 + J_3 \\ \text{s.t} \quad & \mathbf{U}_1,\mathbf{U}_2,\mathbf{V}_1,\mathbf{V}_2,\mathbf{M} \geq \mathbf{0} \end{aligned} \tag{4.14}$$

4.1.3 iNeat: Optimization Algorithm

We first present the proposed algorithm to solve the optimization problem Eq. (4.14). Then, we analyze the algorithm in terms of the correctness, convergence and complexity.

Optimization Algorithm. Since the overall objective function Eq. (4.14) is not jointly convex, we optimize it by block coordinate descent. That is, the objective function is alternatively minimized with respect to one variable group (e.g., $\mathbf{U}_1$) while fixing the others once at a time. For the sake of conciseness, we only show the minimization procedures over $\mathbf{U}_1$

and $\mathbf{M}$ in this section. Other variables such as $\mathbf{V}_1, \mathbf{U}_2, \mathbf{V}_2$ can be solved in a similar way and we omit the details. One can refer to the details in [16].

First, we show the update algorithm over $\mathbf{U}_1$. The gradient of Eq. (4.2) with respect to $\mathbf{U}_1$ is computed by $\frac{\partial J_1}{\partial \mathbf{U}_1} = \mathbf{P}_1 - \mathbf{Q}_1$ where

$$\mathbf{P}_1 = [\mathbf{P}_{\Omega_1} \odot (\mathbf{U}_1\mathbf{V}_1')]\mathbf{V}_1 + \lambda\mathbf{U}_1 \tag{4.15}$$

$$\mathbf{Q}_1 = (\mathbf{P}_{\Omega_1} \odot \mathbf{A}_1)\mathbf{V}_1 \tag{4.16}$$

and $\mathbf{P}_{\Omega_1}(i,j) = 1$ for $(i,j) \in \Omega_1$, otherwise $\mathbf{P}_{\Omega_1}(i,j) = 0$.

As for Eq. (4.11), note that $\mathbf{D}_1 = \text{diag}(\mathbf{U}_1\mathbf{V}_1'\mathbf{1}_1)$ and $\hat{\mathbf{D}}_1 = \text{diag}(\mathbf{1}_1'\mathbf{U}_1\mathbf{V}_1')$ are also in terms of $\mathbf{U}_1$, thus the partial gradient is computed by $\frac{\partial J_2}{\partial \mathbf{U}_1} = \mathbf{P}_2 - \mathbf{Q}_2$ where

$$\begin{aligned}\mathbf{P}_2 = {} & \frac{\gamma}{2}[(\mathbf{U}_1\mathbf{M}\mathbf{U}_2'\mathbf{D}_2\mathbf{U}_2\mathbf{M}') \odot \mathbf{U}_1]\mathbf{1}_{r_1}\mathbf{1}_1'\mathbf{V}_1 + \frac{\gamma}{2}\mathbf{1}_1\mathbf{1}_{r_1}'[(\mathbf{M}\mathbf{U}_2'\hat{\mathbf{D}}_2\mathbf{U}_2\mathbf{M}'\mathbf{U}_1') \odot \mathbf{U}_1']\mathbf{V}_1 \\ & + \gamma(\mathbf{D}_1\mathbf{U}_1\mathbf{M}\mathbf{U}_2'\mathbf{D}_2\mathbf{U}_2\mathbf{M}' + \hat{\mathbf{D}}_1\mathbf{U}_1\mathbf{M}\mathbf{U}_2'\hat{\mathbf{D}}_2\mathbf{U}_2\mathbf{M}')\end{aligned} \tag{4.17}$$

$$\begin{aligned}\mathbf{Q}_2 = {} & \gamma\mathbf{V}_1\mathbf{U}_1'\mathbf{U}_1\mathbf{M}\mathbf{U}_2'\mathbf{U}_2\mathbf{V}_2'\mathbf{U}_2\mathbf{M}' + \gamma\mathbf{U}_1\mathbf{M}\mathbf{U}_2'\mathbf{U}_2\mathbf{V}_2'\mathbf{U}_2\mathbf{M}'\mathbf{U}_1'\mathbf{V}_1 \\ & + \gamma\mathbf{U}_1\mathbf{V}_1'\mathbf{U}_1\mathbf{M}\mathbf{U}_2'\mathbf{V}_2\mathbf{U}_2'\mathbf{U}_2\mathbf{M}'\end{aligned} \tag{4.18}$$

And the gradient of Eq. (4.13) over $\mathbf{U}_1$ is $\frac{\partial J_3}{\partial \mathbf{U}_1} = \mathbf{P}_3 - \mathbf{Q}_3$ where

$$\begin{aligned}\mathbf{P}_3 = {} & 2\beta[\mathbf{P}_{\bar{\Omega}_1} \odot (\mathbf{U}_1\mathbf{M}\mathbf{U}_2'\mathbf{A}_2\mathbf{U}_2\mathbf{M}'\mathbf{U}_1')]\mathbf{U}_1\mathbf{M}\mathbf{U}_2'\mathbf{A}_2\mathbf{U}_2\mathbf{M}' + \beta[\mathbf{P}_{\bar{\Omega}_1} \odot (\mathbf{U}_1\mathbf{V}_1')]\mathbf{V}_1 \\ & + 2\beta\mathbf{A}_1\mathbf{U}_1\mathbf{M}\mathbf{U}_2'[\mathbf{P}_{\bar{\Omega}_2} \odot (\mathbf{U}_2\mathbf{M}'\mathbf{U}_1'\mathbf{A}_1\mathbf{U}_1\mathbf{M}\mathbf{U}_2')]\mathbf{U}_2\mathbf{M}'\end{aligned} \tag{4.19}$$

$$\begin{aligned}\mathbf{Q}_3 = {} & \beta[\mathbf{P}_{\bar{\Omega}_1} \odot (\mathbf{U}_1\mathbf{M}\mathbf{U}_2'\mathbf{A}_2\mathbf{U}_2\mathbf{M}'\mathbf{U}_1')]\mathbf{V}_1 + \beta[\mathbf{P}_{\bar{\Omega}_1} \odot (\mathbf{U}_1\mathbf{V}_1' + \mathbf{V}_1\mathbf{U}_1')]\mathbf{U}_1\mathbf{M}\mathbf{U}_2'\mathbf{A}_2\mathbf{U}_2\mathbf{M}' \\ & + \beta\mathbf{A}_1\mathbf{U}_1\mathbf{M}\mathbf{U}_2'[\mathbf{P}_{\bar{\Omega}_2} \odot (\mathbf{U}_2\mathbf{V}_2' + \mathbf{V}_2\mathbf{U}_2')]\mathbf{U}_2\mathbf{M}'\end{aligned} \tag{4.20}$$

and matrix $\mathbf{P}_{\bar{\Omega}_2}(i,j) = 1$ for any $(i,j) \notin \Omega_2$.

A fixed-point solution of $\frac{\partial J}{\partial \mathbf{U}_1} = \mathbf{0}$ under the non-negativity constraint of $\mathbf{U}_1$ leads to the following multiplicative update rule

$$\mathbf{U}_1(p,q) \leftarrow \mathbf{U}_1(p,q)\sqrt[4]{\frac{\mathbf{Q}_1(p,q) + \mathbf{Q}_2(p,q) + \mathbf{Q}_3(p,q)}{\mathbf{P}_1(p,q) + \mathbf{P}_2(p,q) + \mathbf{P}_3(p,q)}} \tag{4.21}$$

Second, the optimization algorithm over $\mathbf{M}$ is given as below. The gradient of Eq. (4.11) w.r.t $\mathbf{M}$ can be derived as $\frac{\partial J_2}{\mathbf{M}} = \mathbf{P}_4 - \mathbf{Q}_4$ where

$$\mathbf{P}_4 = \gamma\mathbf{U}_1'\mathbf{D}_1\mathbf{U}_1\mathbf{M}\mathbf{U}_2'\mathbf{D}_2\mathbf{U}_2 + \gamma\mathbf{U}_1'\hat{\mathbf{D}}_1\mathbf{U}_1\mathbf{M}\mathbf{U}_2'\hat{\mathbf{D}}_2\mathbf{U}_2 \tag{4.22}$$

$$\mathbf{Q}_4 = \gamma\mathbf{U}_1'\mathbf{U}_1\mathbf{V}_1'\mathbf{U}_1\mathbf{M}\mathbf{U}_2'\mathbf{V}_2\mathbf{U}_2'\mathbf{U}_2 + \gamma\mathbf{U}_1'\mathbf{V}_1\mathbf{U}_1'\mathbf{U}_1\mathbf{M}\mathbf{U}_2'\mathbf{U}_2\mathbf{V}_2'\mathbf{U}_2 \tag{4.23}$$

And the gradient of Eq. (4.13) w.r.t $\mathbf{M}$ is computed by $\frac{\partial J_3}{\mathbf{M}} = \mathbf{P}_5 - \mathbf{Q}_5$ where

$$\begin{aligned}\mathbf{P}_5 &= \beta \mathbf{U}_1' \mathbf{A}_1 \mathbf{U}_1 \mathbf{M} \mathbf{U}_2' [\mathbf{P}_{\bar{\Omega}_2} \odot (\mathbf{U}_2 \mathbf{V}_2' + \mathbf{V}_2 \mathbf{U}_2')] \mathbf{U}_2 \\ &\quad + \beta \mathbf{U}_1' [\mathbf{P}_{\bar{\Omega}_1} \odot (\mathbf{U}_1 \mathbf{V}_1' + \mathbf{V}_1 \mathbf{U}_1')] \mathbf{U}_1 \mathbf{M} \mathbf{U}_2' \mathbf{U}_2' \mathbf{A}_2 \mathbf{U}_2 \qquad (4.24) \\ \mathbf{Q}_5 &= 2\beta \mathbf{U}_1' \mathbf{A}_1 \mathbf{U}_1 \mathbf{M} \mathbf{U}_2' [\mathbf{P}_{\bar{\Omega}_2} \odot (\mathbf{U}_2 \mathbf{M}' \mathbf{U}_1' \mathbf{A}_1 \mathbf{U}_1 \mathbf{M} \mathbf{U}_2')] \mathbf{U}_2 \\ &\quad + 2\beta \mathbf{U}_1' [\mathbf{P}_{\bar{\Omega}_1} \odot (\mathbf{U}_1 \mathbf{M} \mathbf{U}_2' \mathbf{A}_2 \mathbf{U}_2 \mathbf{M}' \mathbf{U}_1')] \mathbf{U}_1 \mathbf{M} \mathbf{U}_2' \mathbf{A}_2 \mathbf{U}_2 \qquad (4.25)\end{aligned}$$

Consequently, the fixed-point solution of $\frac{\partial J}{\partial \mathbf{M}} = \mathbf{0}$ under the non-negative constraint leads to the following update rule

$$\mathbf{M}(p,q) \leftarrow \mathbf{M}(p,q) \sqrt[4]{\frac{\mathbf{Q}_4(p,q) + \mathbf{Q}_5(p,q)}{\mathbf{P}_4(p,q) + \mathbf{P}_5(p,q)}} \qquad (4.26)$$

Initialization. Since the optimization problem in Eq. (4.14) is not a joint convex problem, a good initialization of each variable group could play an important role of obtaining a good final solution. For $\mathbf{U}_1$ and $\mathbf{U}_2$, we initialize them by solving the symmetric non-negative matrix factorization of $\mathbf{A}_1$ and $\mathbf{A}_2$, e.g., minimizing $\|\mathbf{A}_1 - \mathbf{U}_1 \mathbf{U}_1'\|_F^2$ over $\mathbf{U}_1 \geq \mathbf{0}$. Same as [166], we use the following multiplicative update rule to obtain the solution

$$\mathbf{U}_1 \leftarrow \mathbf{U}_1 \odot [1 - \epsilon + \epsilon \frac{\mathbf{A}_1 \mathbf{U}_1}{\mathbf{U}_1 (\mathbf{U}_1' \mathbf{U}_1)}] \qquad (4.27)$$

where ϵ is suggested to be set to 0.5 in practice. Then we set $\mathbf{V}_1 = \mathbf{U}_1$ due to the symmetry of $\mathbf{A}_1$ and initialize $\mathbf{U}_2, \mathbf{V}_2$ similarly. As for the variable $\mathbf{M}$, given the initial $\mathbf{U}_1 = \mathbf{V}_1, \mathbf{U}_2 = \mathbf{V}_2$, we can simplify the computation of Eq. (4.10) and initialize $\mathbf{M}$ as

$$\mathbf{M} = (1 - \alpha) \sum_{k=0}^{K} \alpha^{k+1} (\mathbf{U}_1' \mathbf{D}_1^{-1} \mathbf{U}_1)^k \mathbf{U}_1' \mathbf{D}_1^{-1} \mathbf{L} \mathbf{D}_2^{-1} \mathbf{U}_2 (\mathbf{U}_2' \mathbf{D}_2^{-1} \mathbf{U}_2)^k \qquad (4.28)$$

where the constant K can be set to a relatively large number, e.g., 100.

Overall, the proposed algorithm is summarized in Algorithm 4.1. First, it initializes each variable as line 1. Then, the algorithm alternatively updates each variable group one by one (line 3-7) until it converges or the maximum iteration number t_{max} is reached. The algorithm outputs the complete networks $\mathbf{A}_1^*, \mathbf{A}_2^*$ (line 10), and the alignment matrix $\mathbf{S}$.

Proof and Analysis. In this part, we provide the theoretical analysis of the updating rule of $\mathbf{U}_1$. We first prove that the fixed-point solution of Eq. (4.21) satisfies the KKT condition. Then we analyze its convergence, as well as its time and space complexity. The analyses and

Algorithm 4.1: iNeat: Incomplete Network Alignment.

Input : (1) the adjacency matrices $\mathbf{A}_1,\ \mathbf{A}_2$ of the incomplete networks $\mathcal{G}_1,\ \mathcal{G}_2$, (2) the optional prior alignment preference $\mathbf{L}$, (3) the rank sizes r_1, r_2, (3) the parameters $\alpha, \lambda, \gamma, \beta$, and (4) the maximum iteration number $t_{\max}$.

Output: (1) the alignment matrix $\mathbf{S}$ between $\mathcal{G}_1$ and $\mathcal{G}_2$, and (2) the complete adjacency matrices $\mathbf{A}_1^*, \mathbf{A}_2^*$.

Initialize $\mathbf{U}_1, \mathbf{V}_1, \mathbf{U}_2, \mathbf{V}_2$ by Eq. (4.27), $\mathbf{M}$ by Eq. (4.28), $t = 1$;

while *not converge and* $t \leq t_{max}$ **do**

  Update $\mathbf{U}_1$ by Eq. (4.21);

  Update $\mathbf{V}_1$;

  Update $\mathbf{U}_2$;

  Update $\mathbf{V}_2$;

  Update $\mathbf{M}$ by Eq. (4.26);

  Set $t \leftarrow t + 1$;

end

$\mathbf{A}_1^* = \mathbf{U}_1\mathbf{V}_1'$ and $\mathbf{A}_2^* = \mathbf{U}_2\mathbf{V}_2'$;

$\mathbf{S} = \alpha\mathbf{U}_1\mathbf{M}\mathbf{U}_2' + (1-\alpha)\mathbf{L}$.

proofs for other variables are similar and are omitted for brevity.

Theorem 4.1. Correctness of Eq. (4.21)**.** At convergence, the fixed-point solution of Eq. (4.21) satisfies the KKT condition.

Proof. Let $\mathbf{\Sigma} \in \mathbb{R}^{n_1 \times r_1}$ be the Lagrangian multiplier and the Lagrangian function of Eq. (4.14) be $L(\mathbf{U}_1) = J(\mathbf{U}_1) - \mathrm{Tr}(\mathbf{\Sigma}'\mathbf{U}_1)$. By setting the gradient of L w.r.t $\mathbf{U}_1$ to 0, we obtain

$$\mathbf{\Sigma} = \mathbf{P}_1 + \mathbf{P}_2 + \mathbf{P}_3 - \mathbf{Q}_1 - \mathbf{Q}_2 - \mathbf{Q}_3 \tag{4.29}$$

The KKT complementary condition for the non-negativity of $\mathbf{U}_1$ gives

$$(\mathbf{X}_1 + \mathbf{X}_2 + \mathbf{X}_3 - \mathbf{Y}_1 - \mathbf{Y}_2 - \mathbf{Y}_3) \odot \mathbf{U}_1 = \mathbf{0} \tag{4.30}$$

According to the updating rule Eq. (4.21), at convergence, we have for $\forall p, q$,

$$\mathbf{U}_1(p,q) = \mathbf{U}_1(p,q)\sqrt[4]{\frac{\mathbf{Q}_1 + \mathbf{Q}_2 + \mathbf{Q}_3}{\mathbf{P}_1 + \mathbf{P}_2 + \mathbf{P}_3}} \tag{4.31}$$

which is equivalent to

$$(\mathbf{P}_1 + \mathbf{P}_2 + \mathbf{P}_3 - \mathbf{Q}_1 - \mathbf{Q}_2 - \mathbf{Q}_3) \odot (\mathbf{U}_1)^4 = \mathbf{0} \tag{4.32}$$

Eq. (4.30) and Eq. (4.32) are equivalent, so the KKT condition is satisfied. QED.

Then, we show the convergence of updating $\mathbf{U}_1$ under Eq. (4.21). First, the following lemma gives the auxiliary function for the objective function Eq. (4.14) w.r.t $\mathbf{U}_1$.

Lemma 4.2. Auxiliary function of $J(\mathbf{U}_1)$. Let $J(\mathbf{U}_1)$ denote all the terms in Eq. (4.14) that contains $\mathbf{U}_1$, then the following function $Z(\mathbf{U}_1, \tilde{\mathbf{U}}_1)$

$$\begin{aligned} Z(\mathbf{U}_1, \tilde{\mathbf{U}}_1) = & \underbrace{\frac{1}{4}\sum_{p,q}[(\mathbf{P}_{\Omega_1} \odot (\tilde{\mathbf{U}}_1\mathbf{V}_1'))\mathbf{V}_1](p,q)\frac{\mathbf{U}_1^4(p,q) + \tilde{\mathbf{U}}_1^4(p,q)}{\tilde{\mathbf{U}}_1^3(p,q)}}_{T_1'} \\ & \underbrace{- \sum_{p,q}[(\mathbf{P}_{\Omega_1} \odot \mathbf{A}_1)\mathbf{V}_1](p,q)\tilde{\mathbf{U}}_1(p,q)(1 + \log\frac{\mathbf{U}_1(p,q)}{\tilde{\mathbf{U}}_1(p,q)})}_{T_2'} \\ & + \underbrace{\frac{\lambda}{4}\sum_{p,q}\frac{\mathbf{U}_1^4(p,q) + \tilde{\mathbf{U}}_1^4(p,q)}{\tilde{\mathbf{U}}_1^2(p,q)}}_{T_3'} + \underbrace{\frac{\gamma}{12}\sum_{p,q}\mathbf{Z}_1(p,q)\frac{3\mathbf{U}_1^4(p,q) + \tilde{\mathbf{U}}_1^4(p,q)}{\tilde{\mathbf{U}}_1^3(p,q)}}_{T_4'} \\ & + \gamma T_5' + \underbrace{\frac{\beta}{4}\sum_{p,q}[(\mathbf{P}_{\bar{\Omega}_1} \odot (\tilde{\mathbf{U}}_1\mathbf{V}_1'))\mathbf{V}_1](p,q)\frac{\mathbf{U}_1^4(p,q) + \tilde{\mathbf{U}}_1^4(p,q)}{\tilde{\mathbf{U}}_1^3(p,q)}}_{T_6'} \\ & + \underbrace{\frac{\beta}{2}\sum_{p,q}\mathbf{Z}_2(p,q)\frac{\mathbf{U}_1^4(p,q)}{\tilde{\mathbf{U}}_1^3(p,q)}}_{T_7'} + \underbrace{\frac{\beta}{2}\sum_{p,q}\mathbf{Z}_3(p,q)\frac{\mathbf{U}_1^4(p,q)}{\tilde{\mathbf{U}}_1^3(p,q)}}_{T_8'} + \beta T_9' + \beta T_{10}' \end{aligned} \tag{4.33}$$

where

$$\begin{aligned} \mathbf{Z}_1 = & \frac{1}{2}[(\tilde{\mathbf{U}}_1\mathbf{M}\mathbf{U}_2'\mathbf{D}_2\mathbf{U}_2\mathbf{M}') \odot \tilde{\mathbf{U}}_1]\mathbf{1}_{r_1}\mathbf{1}_1'\mathbf{V}_1 + \frac{1}{2}\mathbf{1}_1\mathbf{1}_{r_1}'[(\mathbf{M}\mathbf{U}_2'\hat{\mathbf{D}}_2\mathbf{U}_2\mathbf{M}'\tilde{\mathbf{U}}_1) \odot \tilde{\mathbf{U}}_1]\mathbf{V}_1 \\ & + \mathrm{diag}(\tilde{\mathbf{U}}_1\mathbf{V}_1'\mathbf{1}_1)\tilde{\mathbf{U}}_1\mathbf{M}\mathbf{U}_2'\mathbf{D}_2\mathbf{U}_2\mathbf{M}' + \mathrm{diag}(\mathbf{V}_1\mathbf{U}_1'\mathbf{1}_1)\tilde{\mathbf{U}}_1\mathbf{M}\mathbf{U}_2'\hat{\mathbf{D}}_2\mathbf{U}_2\mathbf{M}' \end{aligned} \tag{4.34}$$

$$\mathbf{Z}_2 = [\mathbf{P}_{\bar{\Omega}_1} \odot (\tilde{\mathbf{U}}_1\mathbf{M}\mathbf{U}_2'\mathbf{A}_2\mathbf{U}_2\mathbf{M}'\tilde{\mathbf{U}}_1')]\tilde{\mathbf{U}}_1\mathbf{M}\mathbf{U}_2'\mathbf{A}_2\mathbf{U}_2\mathbf{M}' \tag{4.35}$$

$$\mathbf{Z}_3 = \mathbf{A}_1\tilde{\mathbf{U}}_1\mathbf{M}\mathbf{U}_2'[\mathbf{P}_{\bar{\Omega}_2} \odot (\mathbf{U}_2\mathbf{M}'\tilde{\mathbf{U}}_1'\mathbf{A}_1\tilde{\mathbf{U}}_1\mathbf{M}\mathbf{U}_2']\mathbf{U}_2\mathbf{M}' \tag{4.36}$$

$$\begin{aligned} T_5' = & -\sum_{o,p,q,r,s}(\mathbf{M}\mathbf{U}_2'\mathbf{U}_2\mathbf{V}_2'\mathbf{U}_2\mathbf{M}')(o,q)\mathbf{V}_1(p,r)\tilde{\mathbf{U}}_1(s,r)\tilde{\mathbf{U}}_1(s,o) \times \tilde{\mathbf{U}}_1(p,q) \\ & (1 + \log\frac{\mathbf{U}_1(p,q)\mathbf{U}_1(s,r)\mathbf{U}_1(s,o)}{\tilde{\mathbf{U}}_1(p,q)\tilde{\mathbf{U}}_1(s,r)\tilde{\mathbf{U}}_1(s,o)}) \end{aligned} \tag{4.37}$$

$$\begin{aligned} T_9' = & -\sum_{o,p,q,r,s}\mathbf{P}_{\bar{\Omega}_1}(p,o)\mathbf{V}_1(o,q)(\mathbf{M}\mathbf{U}_2'\mathbf{A}_2\mathbf{U}_2\mathbf{M}')(s,r)\tilde{\mathbf{U}}_1(o,s) \times \tilde{\mathbf{U}}_1(p,r)\tilde{\mathbf{U}}_1(p,q) \\ & (1 + \log\frac{\mathbf{U}_1(p,q)\mathbf{U}_1(o,s)\mathbf{U}_1(p,r)}{\tilde{\mathbf{U}}_1(p,q)\tilde{\mathbf{U}}_1(o,s)\tilde{\mathbf{U}}_1(p,r)}) \end{aligned} \tag{4.38}$$

$$T'_{10} = -\sum_{o,p,q,r,s,t} \mathbf{P}_{\bar{\Omega}_2}(p,q)(\mathbf{U}_2\mathbf{V}'_2)(p,q)(\mathbf{U}_2\mathbf{M}')(p,r)\tilde{\mathbf{U}}_1(s,r)\mathbf{A}_1(s,t)\tilde{\mathbf{U}}_1(t,o)(\mathbf{M}\mathbf{U}'_2)(o,q)$$
$$\times (1 + \log \frac{\mathbf{U}_1(s,r)\mathbf{U}_1(t,o)}{\tilde{\mathbf{U}}_1(s,r)\tilde{\mathbf{U}}_1(t,o)}) \tag{4.39}$$

is an auxiliary function of $J(\mathbf{U}_1)$ for any $\mathbf{U}_1, \tilde{\mathbf{U}}_1 \geq \mathbf{0}$ after removing some constant terms such that $Z(\mathbf{U}_1, \tilde{\mathbf{U}}_1) \geq J(\mathbf{U}_1)$ and $Z(\mathbf{U}_1, \mathbf{U}_1) = J(\mathbf{U}_1)$. And it is also a convex function w.r.t $\mathbf{U}_1$ and its global minima is

$$\mathbf{U}_1(p,q) = \tilde{\mathbf{U}}_1(p,q)\sqrt[4]{\frac{\tilde{\mathbf{Q}}_1(p,q) + \tilde{\mathbf{Q}}_2(p,q) + \tilde{\mathbf{Q}}_3(p,q)}{\tilde{\mathbf{P}}_1(p,q) + \tilde{\mathbf{P}}_2(p,q) + \tilde{\mathbf{P}}_3(p,q)}} \tag{4.40}$$

where $\tilde{\mathbf{P}}_i, \tilde{\mathbf{Q}}_i$ are all in terms of $\tilde{\mathbf{U}}_1$ while sharing the same formulas with $\mathbf{P}_i, \mathbf{Q}_i, i = 1, 2, 3$.

Proof. Refer the proof to Appendix B in [16]. QED.

Next, we show the convergence of updating $\mathbf{U}_1$ by Eq. (4.21) in the following theorem.

Theorem 4.2. Convergence of Eq. (4.21)**.** When other variables are fixed, under the updating rule Eq. (4.21), the objective function w.r.t. $\mathbf{U}_1$ monotonically non-increases.

Proof. Denote $\mathbf{U}_1$ at iteration t as $\mathbf{U}_1^{(t)}$. According to Lemma 4.2, the global minima $\mathbf{U}_1^{(t+1)}$ of the auxiliary function is achieved by minimizing $Z(\mathbf{U}_1, \mathbf{U}_1^{(t)})$ over $\mathbf{U}_1$, which leads to

$$Z(\mathbf{U}_1^{(t+1)}, \mathbf{U}_1^{(t)}) \leq Z(\mathbf{U}_1^{(t)}, \mathbf{U}_1^{(t)}) = J(\mathbf{U}_1^{(t)}) \tag{4.41}$$

Besides, based on Lemma 4.2, $J(\mathbf{U}_1^{(t+1)}) \leq Z(\mathbf{U}_1^{(t+1)}, \mathbf{U}_1^{(t)})$ and therefore $J(\mathbf{U}_1^{(t+1)}) \leq J(\mathbf{U}_1^{(t)})$ which means the objective function w.r.t. $\mathbf{U}_1$ is monotonically non-increasing. QED.

The time and space complexities of each updating iteration in Algorithm 4.1 are summarized in Lemma 4.3. Note that by exploring the low-rank structure of the alignment matrix, the time complexity is reduced to *linear*.

Lemma 4.3. Complexity of iNeat. The time complexity of each update iteration in Algorithm 4.1 is $O(nr^2 + \min\{|\bar{\Omega}|, |\Omega|\}r)$, and the space complexity is $O(nr + \min\{|\bar{\Omega}|, |\Omega|\})$ where $n, |\Omega|, |\bar{\Omega}|$ are the number of nodes, the number of observed and unobserved entries in two incomplete networks respectively. And r denotes the rank of networks.

Proof. For time complexity, calculating the term $\mathbf{P}_1, \mathbf{Q}_1, \mathbf{P}_3$ and $\mathbf{Q}_3$ in each iteration has $O(nr^2 + mr + \min\{|\bar{\Omega}|, |\Omega|\}r)$ time complexity and $O(m + nr)$ space complexity. Note that $\mathbf{P}_{\Omega_1} + \mathbf{P}_{\bar{\Omega}_1} = \mathbf{1}_{n_1 \times n_1}$. In this way, for example, $\mathbf{P}_1$ can be computed from either $\mathbf{P}_{\Omega_1}$ or

$\mathbf{P}_{\bar{\Omega}_1}$. Thus, we use $\min\{|\bar{\Omega}|, |\Omega|\}$ for the complexity analysis. For terms $\mathbf{X}_2$ and $\mathbf{Q}_2$, it takes $O(nr^2)$ time complexity and $O(nr)$ space complexity. For other variables $\mathbf{V}_1, \mathbf{U}_2, \mathbf{V}_2$ and $\mathbf{M}$, since the analyses are similar, we omit the analyses for brevity. Overall, we can obtain the time and space complexity in the above lemma. QED.

We remark that the linear complexity is obtained in each updating iteration of Algorithm 4.1. If we carry out line 10-11 in a straightforward way, it will incur an additional $O(n^2)$ cost due to the multiplications between low-rank matrices (e.g., $\mathbf{U}_1\mathbf{V}_1'$, $\mathbf{U}_1\mathbf{M}\mathbf{U}_2'$, etc.) as well as the need to store the potentially dense matrices (e.g., $\mathbf{A}_1^*$, $\mathbf{S}$, etc). To address this issue, we can store the resulting $\mathbf{A}_1^*$, $\mathbf{A}_2^*$ and $\mathbf{S}$ in a compact way by the corresponding low-rank matrices. Then when we access a certain entry of the matrix (e.g. $\mathbf{A}_1^*$), we perform the vector-vector inner product between the corresponding rows of $\mathbf{U}_1$ and $\mathbf{V}_1$.

4.1.4 Experimental Evaluations

In this part, we evaluate the proposed algorithm iNeat in the following two aspects:

- *Effectiveness*: How accurate is our algorithm for aligning incomplete networks? How effective is our algorithm to recover missing edges by leveraging the alignment result?
- *Efficiency*: How fast and scalable is our algorithm?

Experimental Setup. We first introduce the experimental setups as follows[3].
Datasets. We evaluate the proposed algorithm on three types of real-world networks, including the collaboration network, infrastructure network and social networks. The statistics of all the datasets are summarized in Table 4.2.

- *Collaboration Network*: We use the collaboration network in the general relativity and quantum cosmology (Gr-Qc) area from the e-print arXiv [112]. Each node represents an author and there exists an edge if two authors have co-authored at least one paper.
- *Infrastructure Network*: This dataset is a network of Autonomous Systems (AS) inferred from Oregon route-view [112]. In the network, nodes are the routers, and edges represent the peering information among routers.
- *Social Network*: We use the social network collected from Google+ [167]. In the network, nodes are the users and an edge denotes that one user has the other user in

[3]The code can be found in `https://github.com/sizhang92/iNeat-ICDM17`.

Table 4.2: Data statistics.

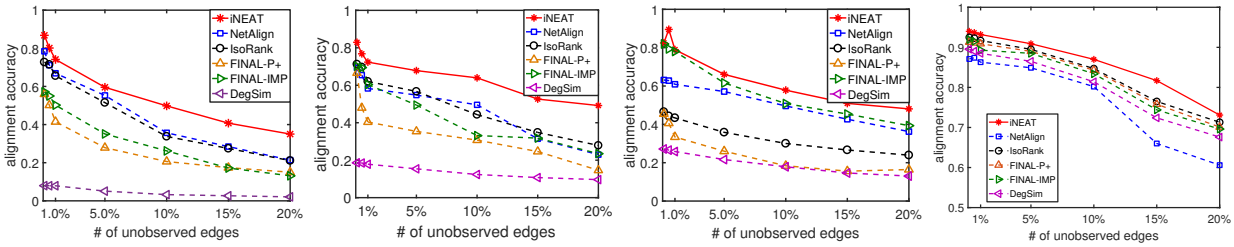
Category	Network	# of Nodes	# of Edges
Collaboration	Gr-Qc	5,241	14,484
Infrastructure	Oregon	7,352	15,665
Social	Google+	23,628	39,194
Social	Youtube	1,134,890	2,987,624
Communication	Phone	1,000	41,191
Communication	Email	1,003	4,627

her circles. We also use the Youtube network [168] where nodes are the Youtube users and edges represent the friendship among users.

- *Contact Networks*: This dataset contains communication networks via different channels. In particular, we aim to align the communication networks via phone (Channel 1) and emails (Channel 2). Each node in both networks represents a person. An edge in Channel 1 network indicates two people contact each other through phone whereas each edge in Channel 2 network represents two people send an email. There are 1,000 common nodes in both networks that are used as the alignment ground-truth.

Based on these datasets except Contact dataset, we construct four pairs of incomplete networks for alignment evaluations by the following steps. For each dataset, we first generate a random permutation matrix and use it to construct the second (permuted) network. Then, in each of these two networks, we remove $0.1\%, 0.5\%, 1\%, 5\%, 10\%, 15\%, 20\%$ of the total number of edges uniformly at random to generate the unobserved edges. For the Contact dataset (Channel 1 and Channel 2), we first compute the edge betweenness score for each edge, which is sum of the fraction of all pairs of shortest paths through the edge [169]. Then we normalize the edge betweenness scores such that they sum to 1, i.e., $\sum_{(u,v)\in\mathcal{G}_1} s_b((u,v)) = 1$ where $s_b((u,v))$ is the edge betweenness score of edge (u,v) and then we use the normalized scores as the probabilities to remove the corresponding edge as an unobserved edge. We run our algorithm and other comparison methods in all the pairs of incomplete networks.
Comparison Methods. We compare iNeat with the following baseline methods.

- *Alignment.* To evaluate the alignment performance of our proposed algorithm iNeat, we compare it with the following prior network alignment algorithms, including (1) NetAlign [5], (2) IsoRank [3], (3) FINAL-P+ [9]. Besides, in order to validate whether alignment and imputation are mutually beneficial from each other, we use the low-rank networks completed solely by Eq. (4.2) as the input networks for FINAL-P+.



(a) Gr-Qc collaboration networks. (b) Oregon infrastructure networks. (c) Google+ social networks. (d) Contact channel 1 vs. channel 2.

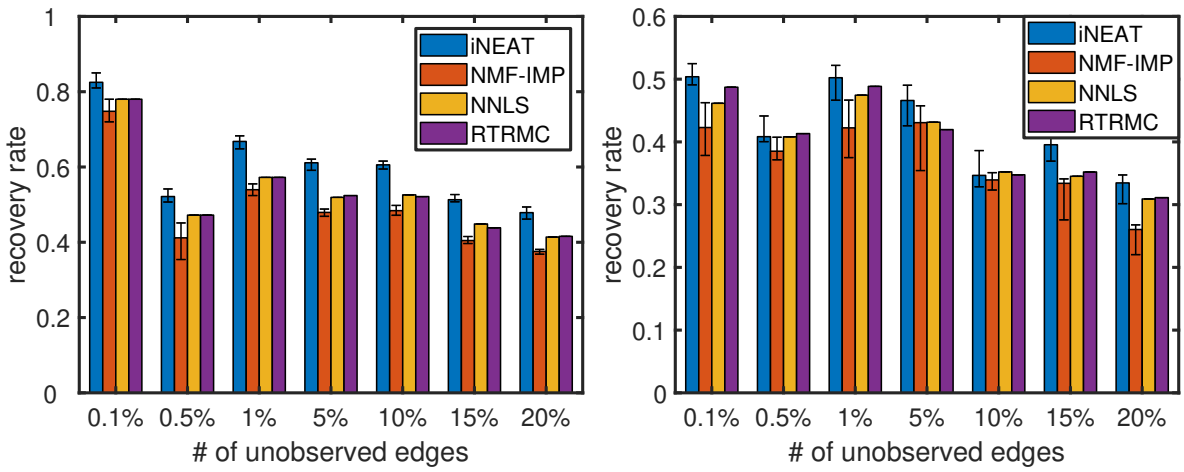
Figure 4.3: (Higher is better.) Alignment accuracy vs. the number of unobserved edges.

We name this method as FINAL-IMP. We also show the alignment results by the degree similarity (DegSim), which is also used as the prior knowledge matrix $\mathbf{L}$.

- *Completion.* To evaluate the completion performance, we compare our algorithm with the existing matrix completion methods which are for the single network completion task, including (1) a matrix factorization method based on Eq. (4.2) (NMF-IMP), (2) an accelerated proximal gradient based nuclear norm minimization method (NNLS) [42], (3) a Riemannian trust-region based matrix completion method (RTRMC) [40].

Machines. All experiments are performed on a Windows machine with four 3.6GHz Intel Cores and 32G RAM. The algorithms are programmed with MATLAB using a single thread.

Effectiveness Analysis. We first evaluate the alignment accuracy with different numbers of unobserved edges in the incomplete networks. We use a heuristic greedy matching algorithm as the post processing step on the alignment matrix to obtain the one-to-one mapping matrix between two input networks, then compute the alignment accuracy with respect to the ground-truth (i.e., the permutation matrix). The results are summarized in Figure 4.3. We have the following observations. First, we observe that iNeat outperforms the baseline methods with different numbers of unobserved edges. To be specific, our method achieves an up to 30% alignment accuracy improvement, compared with the baseline methods that directly align across two incomplete networks (i.e., NetAlign, IsoRank, FINAL-P+). Second, the degree similarity (i.e., $\mathbf{L}$) alone gives a very poor performance on the alignment accuracy, whereas by averaging $\mathbf{L}$ and $\mathbf{U}_1\mathbf{M}\mathbf{U}_2'$, the alignment matrix (i.e., results of iNeat) provides a much better accuracy. This verifies the effectiveness of our strategy combining the low-rank structure of alignment matrix and prior knowledge $\mathbf{L}$. Third, the accuracy of iNeat is higher than that of FINAL-IMP, which indicates that solving the alignment and imputation tasks simultaneously indeed achieves a better performance than the *completion-then-alignment* strategy. Specifically, as Figure 4.3 (a) and Figure 4.3 (b) show, in some



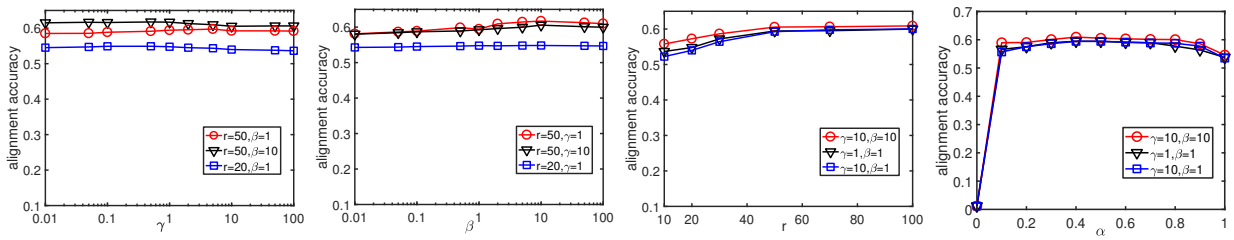
(a) Gr-Qc collaboration network. (b) Google+ social network.

Figure 4.4: (Higher is better.) Recovery rate vs. the number of unobserved edges.

cases, the pure completion may introduce too much noise in the incomplete networks and hence lead to an even worse alignment result than that of other alignment baseline methods (those without performing network completion at all).

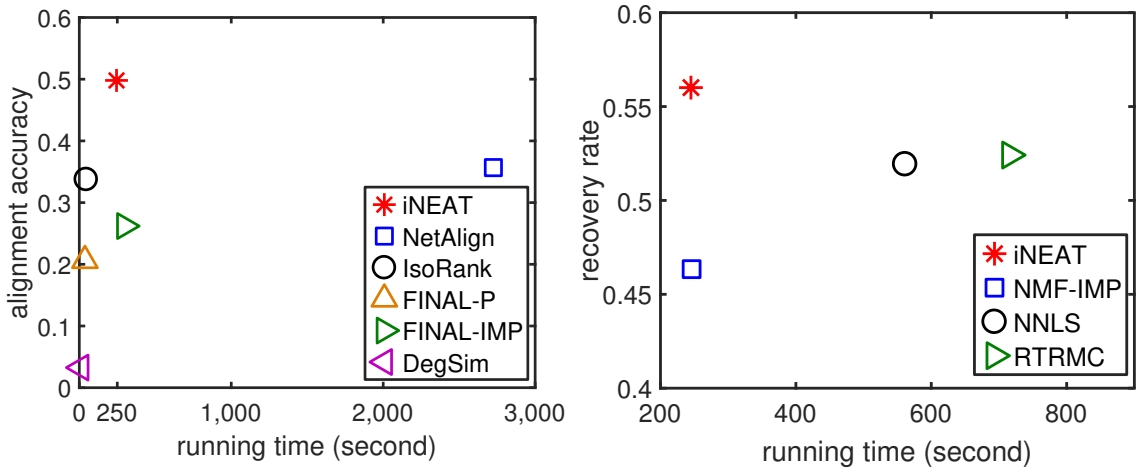
Second, to evaluate the effectiveness of iNeat for network completion, we assume the missing edges are recovered if the corresponding entries of the completed adjacency matrix are larger than a certain threshold (e.g., set to be 0.3 in this work). Then, we calculate the recovery rate over the total number of missing edges. In addition, as the algorithms of network completion may achieve different performance with different initializations, we repeat the algorithms for 20 runs and present the mean edge recovery rates and variances. The results are shown in Figure 4.4. As we can see, iNeat has a higher recovery rate than other baseline methods, indicating that the completion performance is indeed improved by leveraging the alignment across two networks. Besides, the network completion performance of our algorithms are not sensitive to the algorithm initializations. For NNLS and RTRMC baseline methods, we did not observe any variances.

Third, we study how different parameters affect the alignment accuracy. In our experiments, we mainly study three parameters, including (1) γ which controls the importance of alignment task, (2) β which controls the importance of cross-network completion task, and (3) r which is the rank of the complete network. The results are shown in Figure 4.5. As we can see, the alignment accuracy is stable within a wide range of parameter settings. Besides, Figure 4.5 (c) suggests that a relatively small rank might be sufficient to achieve a satisfactory alignment performance. We also observe in Figure 4.5 (d) that (1) by leveraging the combination of both $\mathbf{U}_1\mathbf{M}\mathbf{U}_2'$ and the prior information $\mathbf{L}$ can significantly improve the



(a) Alignment accuracy vs. γ. (b) Alignment accuracy vs. β. (c) Alignment accuracy vs. rank r. (d) Alignment accuracy vs. α.

Figure 4.5: Parameter study on collaboration networks with 5% unobserved edges: study the effect of the parameters γ, β, rank r and α in terms of alignment accuracy.



(a) Balance between running time and alignment accuracy.

(b) Balance between running time and recovery rate.

Figure 4.6: Quality-speed results on the collaboration network with 10% unobserved edges.

alignment performance, and (2) $\alpha = 0.5$ leads to better results in most cases.

Efficiency Analysis. In order to evaluate the trade-off between the effectiveness and efficiency of our method, we measure the quality from two aspects, including the quality of alignment and that of network completion. Here, we show the trade-off results on the collaboration network with 10% unobserved edges in Figure 4.6. As we can see in Figure 4.6 (a), the running time of our method iNeat is slightly higher than IsoRank and FINAL-P+, but it achieves a 15%-25% alignment accuracy improvement across the incomplete networks. Meanwhile, our method is much faster than NetAlign.

Moreover, to evaluate the quality of network completion, note that the running time is the time for completing two incomplete networks. As Figure 4.6 (b) shows, iNeat obtains a better recovery rate and less running time. To be specific, compared with NMF-IMP, iNeat

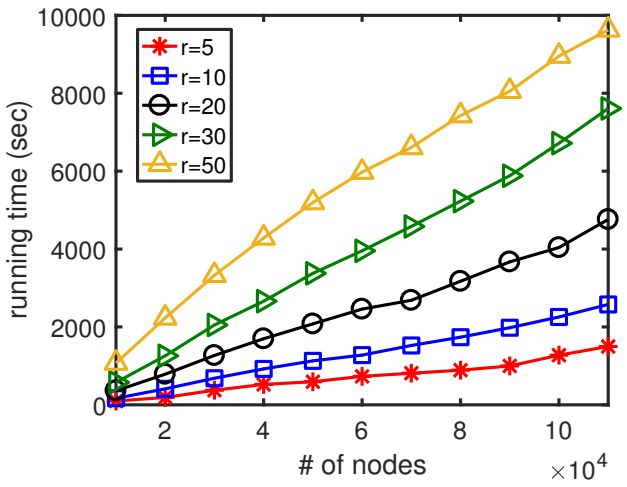


Figure 4.7: Running time vs. the number of nodes in the networks.

can recover 10% more missing edges with a similar running time. Besides, iNeat achieves a slightly better recovery rate and a much faster speed than NNLS and RTRMC.

Scalability. We use the largest dataset (Youtube) to study the scalability of our proposed method iNeat (i.e., running time vs. size of the network). Here, we use the same method to extract and construct several pairs of incomplete subgraphs with different sizes from the entire network. As we can see from Figure 4.7, the running time of the algorithm is *linear* w.r.t. the number of nodes in the networks which is consistent with time complexity analysis.

4.2 ROBUSTNESS ANALYSIS OF NETWORK ALIGNMENT

In this section, we present the robustness analysis of the algorithm FINAL-NE [9] and the analysis on other variants can be easily derived. Given that many real-world networks are often noisy, this analysis assists to perceive how robust our proposed algorithms [12] are to the noise/perturbations. Since the alignment vector $\mathbf{s}$ is the solution to the linear system $(\mathbf{I}-\alpha\tilde{\mathbf{W}})\mathbf{s} = (1-\alpha)\mathbf{l}$, the robustness is equivalent to that of the corresponding linear system. We first provide the lemma as follows.

Lemma 4.4. Robustness of FINAL. If the perturbation on the input networks $\delta = \max\{\|\Delta\mathbf{A}_1\|_F,\ \|\Delta\mathbf{A}_2\|_F\}$ satisfies the following the conditions

$$\delta \leq \sqrt{\min\left\{\frac{\epsilon nA}{\alpha B - \epsilon n^2 A}, \sqrt{\frac{\epsilon nA^2}{2\alpha} + \frac{C^2}{4}} - \frac{C}{2}, \frac{\epsilon n^2 A^2}{\alpha A + \alpha nB + \epsilon n^3 A}\right\} + \frac{D^2}{4}} - \frac{D}{2} \tag{4.42}$$

and

$$\min_i\{d_i\} \leq \frac{\alpha\|\mathbf{E} \odot (\mathbf{A}_2 \otimes \mathbf{A}_1)\|_F}{\epsilon n^2} \tag{4.43}$$

where $d_i = \mathbf{D}(i,i)$, $A = \min_i\{d_i\}$, $B = \|\mathbf{A}_1\|_F\|\mathbf{A}_2\|_F$, $C = \frac{1}{2}(B + \frac{A}{n} - \frac{\epsilon n^2 A}{\alpha})$, $D = \|\mathbf{A}_1\|_F + \|\mathbf{A}_2\|_F$, and $0 < \epsilon < \frac{1-\alpha}{(1+\alpha)n^2}$ is a constant, the relative error of the system due to the perturbation is bounded by

$$\frac{\|\hat{\mathbf{s}} - \mathbf{s}\|_2}{\|\mathbf{s}\|_2} \leq \frac{2\epsilon}{1-r}\kappa_F(\mathbf{I} - \alpha\tilde{\mathbf{W}}) < \frac{2\epsilon(1+\alpha)n^2}{1-\alpha-\epsilon(1+\alpha)n^2} \tag{4.44}$$

where $\kappa_F(\cdot)$ is the condition number in the Frobenius norm.

To prove the above Lemma 4.4, it is equivalent to proving the stability analysis of the linear systems corresponding to the non-perturbed and perturbed linear systems:

$$(\mathbf{I} - \alpha\tilde{\mathbf{W}})\mathbf{s} = (1-\alpha)\mathbf{l} \tag{4.45}$$

$$(\mathbf{I} - \alpha(\tilde{\mathbf{W}} + \Delta\mathbf{W}))\hat{\mathbf{s}} = (1-\alpha)\mathbf{l} \tag{4.46}$$

where $\tilde{\mathbf{W}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{N}[\mathbf{E} \odot (\mathbf{A}_2 \otimes \mathbf{A}_1)]\mathbf{N}\mathbf{D}^{-\frac{1}{2}}$ and $\mathbf{D} = \mathbf{N}[\mathbf{E} \odot (\mathbf{A}_2 \otimes \mathbf{A}_1)]\mathbf{N}\mathbf{1}$. $\Delta\mathbf{W}$ is the perturbation on $\tilde{\mathbf{W}}$. Besides, $\mathbf{s}, \hat{\mathbf{s}}$ are the alignment vectors before and after perturbation, respectively. Before we show the stability analysis of linear systems, we present several propositions to pave the way for our final result.

Denote $\mathbf{P} = \mathbf{E} \odot (\mathbf{A}_2 \otimes \mathbf{A}_1)$ and $\hat{\mathbf{P}} = \mathbf{P} + \Delta\mathbf{P} = \mathbf{E} \odot [(\mathbf{A}_2 + \Delta\mathbf{A}_2) \otimes (\mathbf{A}_1 + \Delta\mathbf{A}_1)]$. Then

$$\Delta\mathbf{P} = \mathbf{E} \odot (\mathbf{A}_2 \otimes \Delta\mathbf{A}_1 + \Delta\mathbf{A}_2 \otimes \mathbf{A}_1 + \Delta\mathbf{A}_2 \otimes \Delta\mathbf{A}_1) \tag{4.47}$$

where $\Delta\mathbf{A}_1$ and $\Delta\mathbf{A}_2$ are the perturbation on the input networks $\mathbf{A}_1$ and $\mathbf{A}_2$, respectively. And denote $d_i = \mathbf{D}(i,i)$ and $\hat{d}_i = \hat{\mathbf{D}}(i,i)$ where $\hat{\mathbf{D}} = \mathbf{N}(\mathbf{P} + \Delta\mathbf{P})\mathbf{N}\mathbf{1}$.

Our first proposition shows $\max_{1\leq i,j\leq n^2}\{|\frac{1}{\sqrt{\hat{d}_i\hat{d}_j}} - \frac{1}{\sqrt{d_i d_j}}|\}$ can be upper bounded.

Proposition 4.1. Let $\Delta_1 = \max_{1\leq i,j\leq n^2}\{|\frac{1}{\sqrt{\hat{d}_i\hat{d}_j}} - \frac{1}{\sqrt{d_i d_j}}|\}$, then Δ_1 is upper-bounded by

$$\Delta_1 \leq \max\left\{\frac{1}{\min_i\{d_i\}} - \frac{1}{\min_i\{d_i\} + n\|\Delta\mathbf{P}\|_F}, \frac{1}{\min_i\{d_i\} - n\|\Delta\mathbf{P}\|_F} - \frac{1}{\min_i\{d_i\}}\right\} \tag{4.48}$$

Proof. Let $C_1 = \max_{1\leq i,j\leq n^2}\{\frac{1}{\sqrt{d_i d_j}} - \frac{1}{\sqrt{\hat{d}_i\hat{d}_j}}\}$ and $C_2 = \max_{1\leq i,j\leq n^2}\{\frac{1}{\sqrt{\hat{d}_i\hat{d}_j}} - \frac{1}{\sqrt{d_i d_j}}\}$. Apparently, $\Delta_1 = \max\{C_1, C_2\}$. By denoting $N_i = \mathbf{N}(i,i)$, since $\hat{d}_i = d_i + N_i\sum_p \Delta\mathbf{P}(i,p)N_p$ and similarly for $\hat{d}_j$, we have the following inequalities:

$$
\begin{aligned}
C_1 &\leq \max_{1\leq i,j\leq n^2}\{\frac{1}{\sqrt{d_i d_j}} - \frac{1}{\sqrt{\left(d_i + N_i\sum_p |\Delta\mathbf{P}(i,p)|N_p\right)\left(d_j + N_j\sum_q |\Delta\mathbf{P}(j,q)|N_q\right)}}\} \\
&\leq \max_{1\leq i,j\leq n^2}\{\frac{1}{\sqrt{d_i d_j}} - \frac{1}{\sqrt{(d_i + \|\Delta\mathbf{P}\|_\infty)(d_j + \|\Delta\mathbf{P}\|_\infty)}}\} \\
&\leq \max_{1\leq i,j\leq n^2}\{\frac{1}{\sqrt{d_i d_j}} - \frac{1}{\sqrt{(d_i + n\|\Delta\mathbf{P}\|_F)(d_j + n\|\Delta\mathbf{P}\|_F)}}\}
\end{aligned} \tag{4.49}
$$

Note that the second line above takes the equality when all perturbations are non-negative, i.e., $\Delta\mathbf{P} \geq 0$. We generalize the r.h.s. of the last inequality above by the function $f(x,y) = \frac{1}{\sqrt{xy}} - \frac{1}{\sqrt{(x+c)(y+c)}}$ where c is any positive constant. And the function $f(x,y)$ has the property that it decreases as either x or y increases. This can be shown by:

$$
\frac{\partial f(x,y)}{\partial x} = \frac{1}{2(x+c)\sqrt{(x+c)(y+c)}} - \frac{1}{2x\sqrt{xy}} < 0 \tag{4.50}
$$

Similarly, we can show $\frac{\partial f(x,y)}{\partial y} < 0$. Thus, C_1 can be further upper bounded by

$$
C_1 \leq \frac{1}{\min_i\{d_i\}} - \frac{1}{\min_i\{d_i\} + n\|\Delta\mathbf{P}\|_F} \tag{4.51}
$$

Next, we give an upperbound for C_2. Similarly to C_1, we have the following inequalities:

$$
\begin{aligned}
C_2 &\leq \max_{1\leq i,j\leq n^2}\{\frac{1}{\sqrt{d_i - N_i\sum_p |\Delta\mathbf{P}(i,p)|N_p}}\frac{1}{\sqrt{d_j - N_j\sum_q |\Delta\mathbf{P}(j,q)|N_q}} - \frac{1}{\sqrt{d_i d_j}}\} \\
&\leq \max_{1\leq i,j\leq n^2}\{\frac{1}{\sqrt{(d_i - \|\Delta\mathbf{P}\|_\infty)(d_j - \|\Delta\mathbf{P}\|_\infty)}} - \frac{1}{\sqrt{d_i d_j}}\} \\
&\leq \max_{1\leq i,j\leq n^2}\{\frac{1}{\sqrt{(d_i - n\|\Delta\mathbf{P}\|_F)(d_j - n\|\Delta\mathbf{P}\|_F)}} - \frac{1}{\sqrt{d_i d_j}}\}
\end{aligned} \tag{4.52}
$$

Note that the second line above takes the equality when all perturbations are non-positive, i.e., $\Delta\mathbf{P} \leq 0$. Then we generalize the last inequality above by the function $g(x,y) = \frac{1}{\sqrt{(x+b)(y+b)}} - \frac{1}{\sqrt{xy}}$ where b is any negative constant. And we can easily show it monotonically decreases as either x or y increases by its derivative $\frac{\partial g(x,y)}{\partial x} < 0$ and $\frac{\partial g(x,y)}{\partial y} < 0$. In this way, C_2 is further upper bounded by

$$
C_2 \leq \frac{1}{\min_i\{d_i\} - n\|\Delta\mathbf{P}\|_F} - \frac{1}{\min_i\{d_i\}} \tag{4.53}
$$

Thus, Δ_1 can be upper bounded by

$$\Delta_1 \leq \max\left\{\frac{1}{\min\limits_i\{d_i\}} - \frac{1}{\min\limits_i\{d_i\} + n\|\Delta\mathbf{P}\|_F},\ \frac{1}{\min\limits_i\{d_i\} - n\|\Delta\mathbf{P}\|_F} - \frac{1}{\min\limits_i\{d_i\}}\right\} \tag{4.54}$$

This completes the proof. QED.

Next, our second lemma shows that $\max\limits_{1\leq i,j\leq n^2}\left\{\frac{1}{\sqrt{\hat{d}_i\hat{d}_j}}\right\}$ can be upper bounded.

Proposition 4.2. Let $\Delta_2 = \max\limits_{1\leq i,j\leq n^2}\left\{\frac{1}{\sqrt{\hat{d}_i\hat{d}_j}}\right\}$, then Δ_2 is upper bounded by

$$\Delta_2 \leq \frac{1}{\min\limits_i\{d_i\} - n\|\Delta\mathbf{P}\|_F} \tag{4.55}$$

Furthermore, when Δ_1 is upper bounded by $\frac{1}{\min\limits_i\{d_i\}} - \frac{1}{\min\limits_i\{d_i\}+n\|\Delta\mathbf{P}\|_F}$, a tighter bound of Δ_2 is

$$\Delta_2 \leq \frac{2}{\min\limits_i\{d_i\}} - \frac{1}{\min\limits_i\{d_i\} + n\|\Delta\mathbf{P}\|_F} \tag{4.56}$$

Proof. Obviously, the value of $\frac{1}{\sqrt{\hat{d}_i\hat{d}_j}}$ increases as either $\hat{d}_i$ or $\hat{d}_j$ decreases. Followed by the conclusion of Eq. (4.53), we know that $\min\limits_i\{\hat{d}_i\} \geq \min\limits_i\{d_i\} - n\|\Delta\mathbf{P}\|_F$. Thus, we have

$$\Delta_2 \leq \frac{1}{\min\limits_i\{\hat{d}_i\}} \leq \frac{1}{\min\limits_i\{d_i\} - n\|\Delta\mathbf{P}\|_F} \tag{4.57}$$

When Δ_1 is upper bounded by $\frac{1}{\min\limits_i\{d_i\}} - \frac{1}{\min\limits_i\{d_i\}+n\|\Delta\mathbf{P}\|_F}$, it means for any indices p, q which satisfy $\hat{d}_p\hat{d}_q \leq d_p d_q$,

$$\frac{1}{\sqrt{\hat{d}_p\hat{d}_q}} - \frac{1}{\sqrt{d_p d_q}} \leq \frac{1}{\min\limits_i\{d_i\}} - \frac{1}{\min\limits_i\{d_i\} + n\|\Delta\mathbf{P}\|_F} \tag{4.58}$$

That is, for these p, q,

$$\begin{aligned}\max_{p,q}\{\frac{1}{\sqrt{\hat{d}_p\hat{d}_q}}\} &\leq \frac{1}{\min\limits_i\{d_i\}} - \frac{1}{\min\limits_i\{d_i\} + n\|\Delta\mathbf{P}\|_F} + \frac{1}{\sqrt{d_p d_q}} \\ &\leq \frac{2}{\min\limits_i\{d_i\}} - \frac{1}{\min\limits_i\{d_i\} + n\|\Delta\mathbf{P}\|_F}\end{aligned} \tag{4.59}$$

For those indices $k,\ l$ where $\hat{d}_k\hat{d}_l \geq d_k d_l$, we have

$$\max_{k,l}\{\frac{1}{\sqrt{\hat{d}_k\hat{d}_l}}\} \leq \max_{k,l}\{\frac{1}{\sqrt{d_k d_l}}\} \leq \frac{1}{\min\limits_i\{d_i\}} \tag{4.60}$$

Since $\frac{1}{\min\limits_i\{d_i\}} \leq \frac{2}{\min\limits_i\{d_i\}} - \frac{1}{\min\limits_i\{d_i\}+n\|\Delta\mathbf{P}\|_F} \leq \frac{1}{\min\limits_i\{d_i\}-n\|\Delta\mathbf{P}\|_F}$, Δ_2 is more tightly bounded by

$$\Delta_2 \leq \frac{2}{\min\limits_i\{d_i\}} - \frac{1}{\min\limits_i\{d_i\}+n\|\Delta\mathbf{P}\|_F} \tag{4.61}$$

And this finishes the proof. QED.

Armed with the above upperbounds, we give the bound for $\delta = \max\{\|\Delta\mathbf{A}_1\|_F, \|\Delta\mathbf{A}_2\|_F\}$ to guarantee $\|\Delta\mathbf{W}\|_F \leq \frac{\epsilon}{\alpha}\|\mathbf{I}-\alpha\tilde{\mathbf{W}}\|_F$.

Proposition 4.3. If $\min\limits_i\{d_i\} \leq \frac{\alpha\|\mathbf{E}\odot(\mathbf{A}_2\otimes\mathbf{A}_1)\|_F}{\epsilon n^2}$, and the maximum perturbation on the input networks δ satisfies

$$\delta \leq \sqrt{\min\left\{\frac{\epsilon nA}{\alpha B-\epsilon n^2 A}, \sqrt{\frac{\epsilon nA^2}{2\alpha}+\frac{C^2}{4}}-\frac{C}{2}, \frac{\epsilon n^2A^2}{\alpha A+\alpha nB+\epsilon n^3 A}\right\}+\frac{D^2}{4}}-\frac{D}{2} \tag{4.62}$$

where $A = \min\limits_i\{d_i\}$, $B = \|\mathbf{A}_1\|_F\|\mathbf{A}_2\|_F$, $C = \frac{1}{2}(B+\frac{A}{n}-\frac{\epsilon n^2 A}{\alpha})$, $D = \|\mathbf{A}_1\|_F + \|\mathbf{A}_2\|_F$, and $\epsilon > 0$ is a constant, then the following is guaranteed:

$$\|\Delta\mathbf{W}\|_F \leq \frac{\epsilon}{\alpha}\|\mathbf{I}-\alpha\tilde{\mathbf{W}}\|_F. \tag{4.63}$$

Proof. We can easily have the following inequalities:

$$\begin{aligned}
\|\Delta\mathbf{W}\|_F &= \|\hat{\mathbf{D}}^{-\frac{1}{2}}\mathbf{N}(\mathbf{P}+\Delta\mathbf{P})\mathbf{N}\hat{\mathbf{D}}^{-\frac{1}{2}} - \mathbf{D}^{-\frac{1}{2}}\mathbf{N}\mathbf{P}\mathbf{N}\mathbf{D}^{-\frac{1}{2}}\|_F \\
&\leq \|\hat{\mathbf{D}}^{-\frac{1}{2}}\mathbf{N}\mathbf{P}\mathbf{N}\hat{\mathbf{D}}^{-\frac{1}{2}} - \mathbf{D}^{-\frac{1}{2}}\mathbf{N}\mathbf{P}\mathbf{N}\mathbf{D}^{-\frac{1}{2}}\|_F + \|\hat{\mathbf{D}}^{-\frac{1}{2}}\mathbf{N}\Delta\mathbf{P}\mathbf{N}\hat{\mathbf{D}}^{-\frac{1}{2}}\|_F \\
&\leq \max_i\{\mathbf{N}(i,i)\}^2(\|\hat{\mathbf{D}}^{-\frac{1}{2}}\mathbf{P}\hat{\mathbf{D}}^{-\frac{1}{2}} - \mathbf{D}^{-\frac{1}{2}}\mathbf{P}\mathbf{D}^{-\frac{1}{2}}\|_F + \|\hat{\mathbf{D}}^{-\frac{1}{2}}\Delta\mathbf{P}\hat{\mathbf{D}}^{-\frac{1}{2}}\|_F) \\
&\leq \|\hat{\mathbf{D}}^{-\frac{1}{2}}\mathbf{P}\hat{\mathbf{D}}^{-\frac{1}{2}} - \mathbf{D}^{-\frac{1}{2}}\mathbf{P}\mathbf{D}^{-\frac{1}{2}}\|_F + \|\hat{\mathbf{D}}^{-\frac{1}{2}}\Delta\mathbf{P}\hat{\mathbf{D}}^{-\frac{1}{2}}\|_F \\
&= \sqrt{\sum_{i,j=1}^{n^2}(\frac{1}{\sqrt{\hat{d}_i\hat{d}_j}}-\frac{1}{\sqrt{d_i d_j}})^2(\mathbf{P}(i,j))^2} + \sqrt{\sum_{i,j=1}^{n^2}(\frac{1}{\sqrt{\hat{d}_i\hat{d}_j}})^2(\Delta\mathbf{P}(i,j))^2} \\
&\leq \Delta_1\|\mathbf{P}\|_F + \Delta_2\|\Delta\mathbf{P}\|_F
\end{aligned} \tag{4.64}$$

Meanwhile, $\|\mathbf{I}-\alpha\tilde{\mathbf{W}}\|_F = \sqrt{n^4+\alpha^2\|\tilde{\mathbf{W}}\|_F^2} \geq n^2$. In this way, to guarantee $\|\Delta\mathbf{W}\|_F \leq$

$\frac{\epsilon}{\alpha}\|\mathbf{I}-\alpha\tilde{\mathbf{W}}\|_F$, we need

$$\Delta_2\|\Delta\mathbf{P}\|_F \leq \frac{\epsilon n^2}{\alpha} - \Delta_1\|\mathbf{P}\|_F. \tag{4.65}$$

Next, we divide the following proof into two cases, depending on the upper bound of Δ_1.

Case 1. Based on Eq. (4.48) and Eq. (4.56), when Δ_1 is upper bounded by $\frac{1}{\min_i\{d_i\}} - \frac{1}{\min_i\{d_i\}+n\|\Delta\mathbf{P}\|_F}$, Δ_2 is upper bounded by $\frac{2}{\min_i\{d_i\}} - \frac{1}{\min_i\{d_i\}+n\|\Delta\mathbf{P}\|_F}$. In order to make sense on $\|\Delta\mathbf{P}\|_F$, we first need to satisfy $\frac{\epsilon n^2}{\alpha} - \Delta_1\|\mathbf{P}\|_F \geq 0$. Thus, we have

$$\Delta_1 \leq \frac{1}{\min_i\{d_i\}} - \frac{1}{\min_i\{d_i\}+n\|\Delta\mathbf{P}\|_F} \leq \frac{\epsilon n^2}{\alpha\|\mathbf{E}\odot(\mathbf{A}_2\otimes\mathbf{A}_1)\|_F} \tag{4.66}$$

By solving the above inequality, we have

$$\|\Delta\mathbf{P}\|_F \leq \frac{\epsilon n \min_i\{d_i\}}{\alpha\|\mathbf{E}\odot(\mathbf{A}_2\otimes\mathbf{A}_1)\|_F - \epsilon n^2\min_i\{d_i\}} \tag{4.67}$$

$$\min_i\{d_i\} \leq \frac{\alpha\|\mathbf{E}\odot(\mathbf{A}_2\otimes\mathbf{A}_1)\|_F}{\epsilon n^2} \tag{4.68}$$

Among others, Eq. (4.68) is to make the r.h.s. of Eq. (4.67) greater than or equal to 0. Moreover, by substituting the upperbound of Δ_1 and Δ_2 (i.e., Eq. (4.48) and Eq. (4.56)) into Eq. (4.65), we have

$$\begin{aligned}\frac{\epsilon n^2}{\alpha} \geq &(\frac{1}{\min_i\{d_i\}} - \frac{1}{\min_i\{d_i\}+n\|\Delta\mathbf{P}\|_F})\|\mathbf{A}_1\|_F\|\mathbf{A}_2\|_F \\ &+ (\frac{2}{\min_i\{d_i\}} - \frac{1}{\min_i\{d_i\}+n\|\Delta\mathbf{P}\|_F})\|\Delta\mathbf{P}\|_F\end{aligned} \tag{4.69}$$

which can be solved by

$$\|\Delta\mathbf{P}\|_F \leq \sqrt{\frac{\epsilon n \min_i\{d_i\}^2}{\alpha} + \frac{C^2}{4}} - \frac{C}{2} \tag{4.70}$$

where $C = \frac{1}{2}(\|\mathbf{A}_1\|_F\|\mathbf{A}_2\|_F + \frac{\min_i\{d_i\}}{n} - \frac{\epsilon n^2\min_i\{d_i\}}{\alpha})$.

Then based on Eq. (4.47), since $\|\Delta\mathbf{P}\|_F \leq \delta^2 + (\|\mathbf{A}_1\|_F + \|\mathbf{A}_2\|_F)\delta$ combined with Eq. (4.67) and Eq. (4.70), we have

$$\delta \leq \min\left\{\sqrt{\frac{\epsilon nA}{\alpha B - \epsilon n^2 A} + \frac{D^2}{4}},\ \sqrt{\sqrt{\frac{\epsilon nA^2}{\alpha} + \frac{C^2}{4}} - \frac{C}{2} + \frac{D^2}{4}}\right\} - \frac{D}{2} \tag{4.71}$$

where $A = \min_i\{d_i\}$, $B = \|\mathbf{A}_1\|_F\|\mathbf{A}_2\|_F$ and $D = (\|\mathbf{A}_1\|_F + \|\mathbf{A}_2\|_F)\ /\ 2$.

Case 2. When Δ_1 is upper bounded by $\frac{1}{\min_i\{d_i\}-n\|\Delta\mathbf{P}\|_F} - \frac{1}{\min_i\{d_i\}}$, by substituting Eq. (4.48) and Eq. (4.55) into Eq. (4.65), we have the following inequality:

$$\frac{\epsilon n^2}{\alpha} \geq (\frac{1}{\min_i\{d_i\} - n\|\Delta\mathbf{P}\|_F} - \frac{1}{\min_i\{d_i\}})\|\mathbf{A}_1\|_F\|\mathbf{A}_2\|_F + \frac{\|\Delta\mathbf{P}\|_F}{\min_i\{d_i\} - n\|\Delta\mathbf{P}\|_F} \tag{4.72}$$

which can be solved by

$$\|\Delta\mathbf{P}\|_F \leq \frac{\epsilon n^2 A^2}{\alpha n B + \alpha A + \epsilon n^3 A} \tag{4.73}$$

Moreover, since we need to guarantee $\Delta_1 \leq \frac{\epsilon n^2}{\alpha\|\mathbf{E}\odot(\mathbf{A}_2\otimes\mathbf{A}_1)\|_F}$, we have another bound for $\|\Delta\mathbf{P}\|_F$ in this case as below:

$$\|\Delta\mathbf{P}\|_F \leq \frac{\epsilon n A^2}{\epsilon n^2 A + \alpha\|\mathbf{E}\odot(\mathbf{A}_2\otimes\mathbf{A}_1)\|_F} \tag{4.74}$$

Because the r.h.s in Eq. (4.74) is greater than that in Eq. (4.73), the upper bound of Eq. (4.73) naturally guarantee $\|\Delta\mathbf{P}\|_F$ has a meaningful solution. Then, similarly, given $\|\Delta\mathbf{P}\|_F \leq \delta^2 + (\|\mathbf{A}_1\|_F + \|\mathbf{A}_2\|_F)\delta$, we obtain the upper bound of δ in this case as following:

$$\delta \leq \sqrt{\frac{\epsilon n^2 A^2}{\alpha n B + \alpha A + \epsilon n^3 A} + \frac{D^2}{4}} - \frac{D}{2} \tag{4.75}$$

In word, by combining Eq. (4.71) and Eq. (4.75), we have

$$\delta \leq \sqrt{\min\left\{\frac{\epsilon n A}{\alpha B - \epsilon n^2 A}, \sqrt{\frac{\epsilon n A^2}{2\alpha} + \frac{C^2}{4}} - \frac{C}{2}, \frac{\epsilon n^2 A^2}{\alpha A + \alpha n B + \epsilon n^3 A}\right\} + \frac{D^2}{4}} - \frac{D}{2} \tag{4.76}$$

which finishes the proof. QED.

With all these ingredients, we now prove Lemma 4.4 by using the stability analysis of the linear systems in Eq. (4.45). Specifically, based on the well-known sensitivity analysis of linear system [170], if the linear systems in Eq. (4.45) satisfy that: (1) $\alpha\|\Delta\mathbf{W}\|_F \leq \epsilon\|\mathbf{I} - \alpha\tilde{\mathbf{W}}\|_F$, and (2) $r = \epsilon\kappa_F(\mathbf{I} - \alpha\tilde{\mathbf{W}}) < 1$, then

$$\frac{\|\hat{\mathbf{s}} - \mathbf{s}\|_F}{\|\mathbf{s}\|_F} \leq \frac{2\epsilon}{1-r}\kappa_F(\mathbf{I} - \alpha\tilde{\mathbf{W}}) \tag{4.77}$$

The first condition can be satisfied by Eq. (4.42) and Eq. (4.68). For the second condition,

given that

$$\kappa_F(\mathbf{I} - \alpha\tilde{\mathbf{W}}) \leq n^2\kappa_2(\mathbf{I} - \alpha\tilde{\mathbf{W}}) < \frac{(1+\alpha)n^2}{1-\alpha} \tag{4.78}$$

we need the constant $\epsilon < \frac{1-\alpha}{(1+\alpha)n^2}$. Then we can have

$$\frac{\|\hat{\mathbf{s}} - \mathbf{s}\|_F}{\|\mathbf{s}\|_F} \leq \frac{2\epsilon}{1-r}\kappa_F(\mathbf{I} - \alpha\tilde{\mathbf{W}}) < \frac{2\epsilon(1+\alpha)n^2}{1-\alpha-\epsilon(1+\alpha)n^2} \tag{4.79}$$

which finishes the proof.

CHAPTER 5: VELOCITY IN BIG NETWORK ALIGNMENT

Real-world networks are often dynamically changing over time. In addition to the structural and attribute information accompanied with networks, the dynamic changes naturally embrace the extra temporal information that may help with network alignment. To align multiple networks, we model networks by a classic multi-graph model, i.e., network of networks such that the alignment inference across multiple networks can be reformulated as a common node prediction problem across domain-specific networks. To this end, we address dynamic network alignment by learning node representations on dynamic network of networks, based on which we further infer node alignments. In this chapter, we propose a novel node representation learning model on dynamic network of networks and evaluate it in the task of dynamic network alignment.

5.1 MOTIVATIONS

Multiple networks, each of which models the complicated relationships among entities in a certain domain, naturally emerge in a plethora of applications, such as social analysis, bioinformatics and team optimization. Despite the capability of capturing the relational information within each domain of network, the classic multi-network models (e.g., multiplex networks) might fail to model the intrinsic graph-level relationships among networks. Network of networks (NoN) model [103, 171], which composes of a main network whose nodes represent a set of domain-specific networks and edges indicate the relations among them, embraces the abilities to leverage both the domain-specific structural information of nodes (i.e., at fine granularity) and the structure of main network (i.e., at coarse granularity). For example, in Figure 5.1, each project-oriented team can be considered as a domain-specific network that models the collaborations among the team members, while different teams are also connected by a main network indicating to what extent different projects are related.

Graph convolutional networks (GCN) have been extensively studied to learn node representations that incorporate node structural and attribute information. Most of the existing GCN models focus on a single network by aggregating node hidden representations from the neighborhood [122, 123, 172], while other methods learn node representations on multiplex networks [173, 174] by integrating multiple layers/views through the set of common nodes. However, these methods inevitably ignore the graph-level relations (e.g., graph similarities) among multiple networks. On the other hand, graph convolutional networks can be used to learn graph similarities among networks by modeling graph-graph interactions with neural

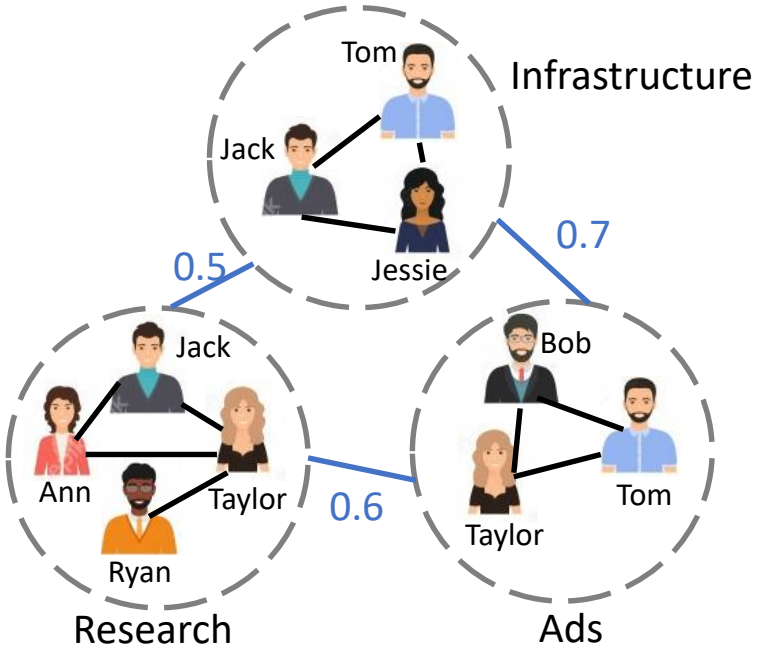


Figure 5.1: An example of network of team networks.

tensor networks on graph representations [175], while overlooking the set of common nodes across different networks.

In the meanwhile, as networks often evolve over time, it is of key importance to leverage the temporal information to learn dynamic node representations. Classic dynamic network embedding methods encode the temporal information by either modeling snapshot networks [70, 71] or explicitly modeling temporal evolution [176, 177]. With the advances of deep learning, neural networks are designed to capture the dynamic changes of network structures, including recurrent neural networks based models [73, 178, 179] and attention based models [75, 180]. Specifically, Pareja et al. propose to encode network structural changes by treating the parameter matrix of GCN as a dynamical system [73]. Sankar at al. propose an attentive aggregation along the temporal dimension to capture the temporal dependence across snapshots [75]. These methods can be used to model the dynamics such as node/edge insertions/deletions in a single network.

Nevertheless, these dynamic graph neural networks still bear the following limitations. First (*domain correlations*), different domain-specific networks often correlate with each other through the edges of main network at the macro level and the set of common nodes shared across different domains at the micro level. However, it is unclear how to encode such correlations into node representation learning even in the static setting. Second (*complex dynamics*), the dynamic evolution underlying each network is sophisticated and the correlation among different domain-specific networks would make the dynamic patterns underlying

network of networks more complex. For example, nodes shared across different domains may be added or deleted over time, which might influence the representation learning of other domains by imposing more or less cross-network correlations respectively. Yet, it is nontrivial to assimilate such dynamics into node representations in dynamic network of networks.

In this work, we address these limitations by developing a dynamic graph neural networks model DraNoN on dynamic network of networks. Specifically, we design a message passing scheme in the static setting which naturally embodies both within-network and cross-network consistency behind node representations inspired by the *predict-then-propagate* strategy. To model the complex dynamics, the key ideas are two-fold. First, we apply a gated recurrent unit (GRU) to rivet on the dynamics behind the sequence of common nodes' representations, whereas the dynamics of non-overlapped nodes is only captured by propagating the hidden representations of common nodes. To better model the dynamics, we further utilize a self-attention based model along temporal dimension such that the dependence with historical representations can be captured. The main contributions are summarized as follows:

- *Problem definition.* To our best knowledge, we are the first to study the representation learning on a dynamic network of networks.
- *Model design.* We first develop DraNoN-S to learn node representations on static network of networks. Then we extend it to DraNoN by applying a GRU and self-attention to model the complex dynamics behind the dynamic network of networks.
- *Experiments.* We evaluate the designed model DraNoN in the task of dynamic network alignment. Extensive experiments demonstrate the significant improvements of DraNoN compared to the state-of-the-arts.

5.2 PROBLEM DEFINITION

Table 5.1 summarizes the main notations. We use bold uppercase letters for matrices (e.g., $\mathbf{A}$), bold lowercase letters for vectors (e.g., $\mathbf{x}$), lowercase plain letters (e.g., α) for scalars and uppercase calligraphic letters (e.g., $\mathcal{I}$) for sets. We use $\mathbf{A}(i, j)$ to denote the entry at the intersection of the i-th row and the j-th column of the matrix $\mathbf{A}$. We denote the transpose by a superscript prime (e.g., $\mathbf{x}'$ as the transpose of $\mathbf{x}$).

5.2.1 Representation Learning on Dynamic NoN

We denote an undirected weighted network by $\mathcal{G} = \{\mathcal{V}, \mathbf{A}\}$ where $\mathcal{V}$ represent the node set of network $\mathcal{G}$ and $\mathbf{A}$ is the adjacency matrix of $\mathcal{G}$ where $\mathbf{A}(i, j) > 0$ indicates the weight

Table 5.1: Symbols and notations.

Symbols	Definitions
$\mathcal{G}$	dynamic network of networks
$\mathcal{G}_0^t$	main network at snapshot t
$\mathbf{A}_0^t$	adjacency matrix of main network at snapshot t
$\mathcal{G}_l^t$	domain-specific network of the l-th domain at snapshot t
$\mathbf{A}_l^t$	adjacency matrix of $\mathcal{G}_l^t$ at snapshot t
$\mathbf{x}_{l,i}^t$	node representation of node-i in $\mathcal{G}_l^t$
$\mathcal{I}_{kl}^t$	the common nodes between $\mathcal{G}_k^t$ and $\mathcal{G}_l^t$
g	number of domain-specific networks
n_l^t	number of nodes in $\mathcal{G}_l^t$
n_c^t	number of common nodes among all domain networks
n^t	number of nodes in all networks at snapshot t
$[\cdot\|\|\cdot]$	vertical concatenation
$\odot$	Hadamard product

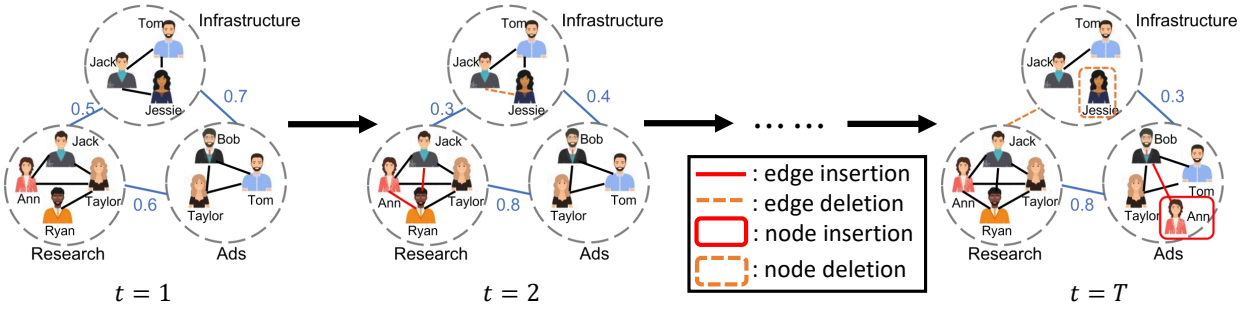


Figure 5.2: An illustrative example of dynamic network of networks with T snapshots. Edge insertions/deletions occur in both domain-specific networks and main network, while node insertions/deletions only happen in the domain-specific networks.

of edge (i, j). A network of networks is composed of a main network whose nodes represent a set of domain-specific networks. We provide the definition of network of networks [103].

Definition 5.1. Network of networks. Given a $g \times g$ main network $\mathcal{G}_0 = \{\mathcal{V}_0, \mathbf{A}_0\}$, each node represents a unique domain-specific network, i.e., $\mathcal{V}_0 = \{\mathcal{G}_1, \cdots, \mathcal{G}_g\}$ where $\mathcal{G}_l = \{\mathcal{V}_l, \mathbf{A}_l\}$, $l = 1, \cdots, g$. Here, $\mathbf{A}_0(k, l)$ measures the relation between $\mathcal{G}_k$ and $\mathcal{G}_l$. For domain-specific network $\mathcal{G}_l$, $\mathbf{A}_l(i, j)$ indicates the weight of edge (i, j).

In this work, we consider a network of networks whose domain-specific networks have a set of common nodes. We define the common nodes between $\mathcal{G}_k$ and $\mathcal{G}_l$ by $\mathcal{I}_{kl}$, i.e., $\mathcal{I}_{kl} = \mathcal{V}_k \cap \mathcal{V}_l$, $k, l = 1, \cdots, g$. Given the fact that many networks often evolve over time, dynamic changes (e.g., nodes/edges insertions/deletions) also happen in the network of networks. Note that we do not consider node insertion and deletion in the main network. In other

words, the main network has g nodes throughout the time span. Take Figure 5.2 as an example. Structural changes occur on the main network over time, such as the changes on edge weights of main network at $t = 2$ and the edge deletion (denoted by brown dashed line) between domains Infrastructure and Research at $t = T$. As for domain-specific networks, edges are added (denoted by red solid lines) in the Research team network and an edge is removed in the Infrastructure team network at $t = 2$. At the last snapshot $t = T$, Jessie leaves the Infrastructure team and Ann joins in the Ads team due to project demands. More formally, we define the dynamic network of networks as below.

Definition 5.2. Dynamic network of networks. Given a set of $g \times g$ main networks $\mathcal{G} = \{\mathcal{G}_0^1, \cdots, \mathcal{G}_0^T\}$ where $\mathcal{G}_0^t = \{\mathcal{V}_0^t, \mathbf{A}_0^t\}$ represents the main network at snapshot t, nodes of main network $\mathcal{G}_0^t$ denote the domain-specific networks at snapshot t, i.e., $\mathcal{V}_0^t = \{\mathcal{G}_1^t, \cdots, \mathcal{G}_g^t\}$ where each domain-specific network is $\mathcal{G}_l^t = \{\mathcal{V}_l^t, \mathbf{A}_l^t\},\ l = 1, \cdots, g$. The adjacency matrix $\mathbf{A}_0^t(k, l)$ indicates the correlation between domain-specific networks $\mathcal{G}_k^t$ and $\mathcal{G}_l^t$, and $\mathbf{A}_l^t(i, j) > 0$ represents the weight of edge (i, j) at snapshot t.

Here we use $\mathcal{G}$ to denote dynamic network of networks for clarity. In addition, $\mathcal{I}_{kl}^t$ denotes common nodes between $\mathcal{G}_k^t$ and $\mathcal{G}_l^t$ (i.e., $\mathcal{I}_{kl}^t = \mathcal{V}_k^t \cap \mathcal{V}_l^t$) and $|\mathcal{V}_l^t| = n_l^t$ denotes the number of nodes in the domain-specific network $\mathcal{G}_l$ at snapshot t. Given the above notations and definitions, our goal is to learn dynamic node representations in the domain-specific networks. Formally, we define the dynamic NoN representation learning problem as follows.

Problem 5.1. Representation Learning on Dynamic NoN.

Given: the dynamic network of networks $\mathcal{G} = \{\mathcal{G}_0^1, \cdots, \mathcal{G}_0^T\}$,

Output: node embeddings $\mathbf{X}_l^t \in \mathbb{R}^{n_l^t \times d},\ \forall l = 1, \cdots, g$ where $\mathbf{X}_l^t(i, :) = \mathbf{x}_{l,i}^t$ represents the representation of node-i in the domain-specific network $\mathcal{G}_l^t$ at snapshot $t = 1, \cdots, T$.

5.2.2 Preliminaries: Ranking on NoN

Given a set of query vectors $\mathbf{e}_l,\ l = 1, \cdots, g$, a classic ranking algorithm on network of networks is attributed to CrossRank [103] whose key ideas are to (1) minimize the difference of ranking scores between neighboring nodes in each domain-specific network (i.e., *within-network smoothness*), (2) minimize the difference between the ranking vectors and the corresponding query vectors to encode the personalization of the query nodes (i.e., *query personalization*), and (3) minimize the ranking score differences of the common nodes that are shared by two highly-correlated domain-specific networks (i.e., *cross-network consistency*). Mathematically, these are formulated into a single optimization problem as

$$J(\mathbf{r}_1, \cdots, \mathbf{r}_g) = c \sum_{l=1}^{g} \mathbf{r}_l'(\mathbf{I}_{n_l} - \tilde{\mathbf{A}}_l)\mathbf{r}_l + (1-c) \sum_{l=1}^{g} \|\mathbf{r}_l - \mathbf{e}_l\|_2^2 + a \sum_{l=1}^{g} \sum_{k=1}^{g} \| \frac{\mathbf{r}_l(\mathcal{I}_{kl})}{\sqrt{\mathbf{D}_0(l,l)}} - \frac{\mathbf{r}_k(\mathcal{I}_{kl})}{\sqrt{\mathbf{D}_0(k,k)}} \|_2^2 \mathbf{A}_0(k,l) \tag{5.1}$$

where $\mathbf{r}_l$ denotes the ranking scores of nodes in the domain-specific network $\mathcal{G}_l$, $\tilde{\mathbf{A}}_l$ denotes the symmetrical normalization of $\mathbf{A}_l$, and $\mathbf{D}_0$ is the degree matrix of the main network $\mathcal{G}_0$, i.e., $\mathbf{D}_0(l,l) = \sum_{k=1}^{g} \mathbf{A}_0(l,k)$. $\mathcal{I}_{kl} = \mathcal{V}_k \cap \mathcal{V}_l$ denotes the common nodes shared between $\mathcal{G}_k$ and $\mathcal{G}_l$. In addition, a, c control the importance of different terms. By taking the derivative of Eq. (5.1) to zero, a fixed-point solution is proposed to minimize Eq. (5.1) [103]

$$\mathbf{r} = (\frac{c}{1+2a}\tilde{\mathbf{A}} + \frac{2a}{1+2a}\tilde{\mathbf{Y}})\mathbf{r} + \frac{1-c}{1+2a}\mathbf{e} \tag{5.2}$$

where $\mathbf{r} = [\mathbf{r}_1 \| \cdots \| \mathbf{r}_g]$ and $\mathbf{e} = [\mathbf{e}_1 \| \cdots \| \mathbf{e}_g]$. $\tilde{\mathbf{A}} = \text{diag}(\tilde{\mathbf{A}}_1, \cdots, \tilde{\mathbf{A}}_g)$ is a $g \times g$ block diagonal matrix. By letting $\mathbf{O}$ be a block matrix whose (k,l)-th block is $\mathbf{A}_0(k,l)\mathbf{O}_{kl}$ where the indicator matrix $\mathbf{O}_{kl}(i,j) = 1$ if node-i in $\mathcal{G}_k$ represents the same node as node-j in $\mathcal{G}_l$, the matrix $\tilde{\mathbf{Y}}$ is the symmetric normalization of $\mathbf{Y} = \mathbf{O} + \mathbf{D}_T$ where $\mathbf{D}_T = \text{diag}(\mathbf{D}_0(1,1)\mathbf{I}_{n_1}, \cdots, \mathbf{D}_0(g,g)\mathbf{I}_{n_g}) - \mathbf{D}_O$ and $\mathbf{D}_O$ is the degree matrix of $\mathbf{O}$. This iterative approach converges to a closed-form solution

$$\mathbf{r} = \frac{1-c}{1+2a}(\mathbf{I} - \frac{c}{1+2a}\tilde{\mathbf{A}} - \frac{2a}{1+2a}\tilde{\mathbf{Y}})^{-1}\mathbf{e}. \tag{5.3}$$

5.3 THE DESIGNED DRANON MODEL

5.3.1 Model Overview

The overall architecture of the model is shown in Figure 5.3. The key challenges of learning node representations on dynamic network of networks include: (1) how to design graph convolutional network model that can learn informative node representations on static NoN, and (2) how to extend the static model to the dynamic setting such that the whole model can effectively capture he complex dynamic patterns underlying the dynamic NoN. To address the first challenge, we aim to design a message passing scheme that can preserve the within-network smoothness in the node's local neighborhood and the cross-network consistency. To be more specific, inspired by the *predict-then-propagate* strategy for single network [181], we

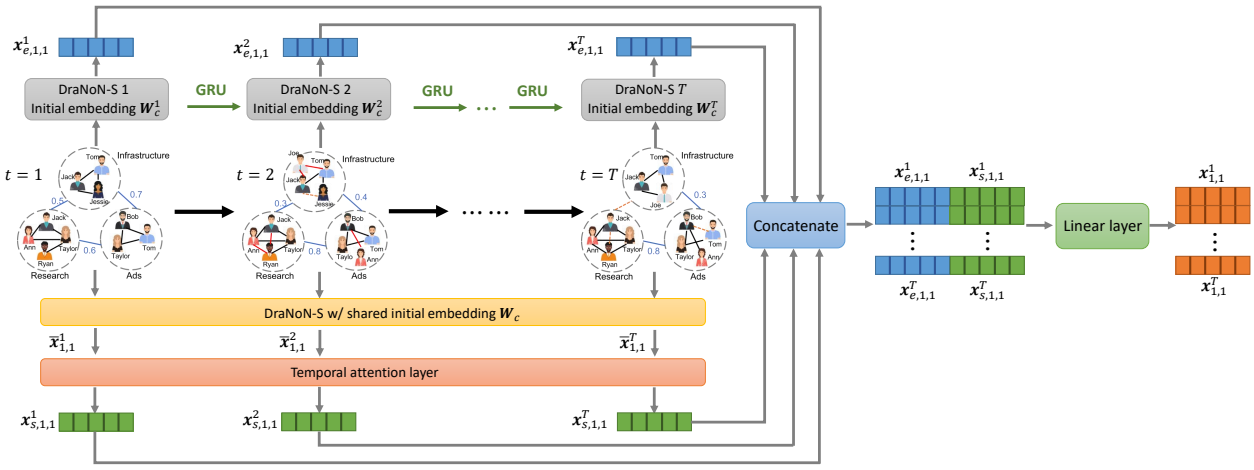


Figure 5.3: Model overview of DraNoN. For clarity, we denote the node representations of Tom in Infrastructure team network at snapshot t by $\mathbf{x}_{1,1}^t$ as an example.

compute the influence matrix between any two nodes in all domain-specific networks based on the closed-form solution of the ranking scores (i.e., Eq. (5.3)). This influence matrix can be viewed as a one-time propagation matrix. In the prediction step, different from [181] which learns a neural network that generates initial node classification by input node features, we instead propose to learn the initial node representations of the common nodes shared across domains in our model. Then to efficiently compute the propagation procedure, we approximate the one-time propagation (i.e., Eq. (5.3)) by a P-step iterative propagation.

As aforementioned, the dynamics of the common nodes (e.g., $\mathcal{I}_{kl}$) play a crucial role in correlating different domain-specific networks and adjusting the cross-network consistency of node representations. In this way, we first resort to a GRU based architecture to effectively model the dynamics behind the common nodes. Moreover, we propose to use the self-attention mechanism along the temporal dimension to capture the dependence on node's historical representations.

5.3.2 DraNoN-S: Embedding on Static NoN

Let us first revisit the closed-form solution to the ranking problem on NoN. Given a query node-i in $\mathcal{G}_l$ (i.e., $i \in \mathcal{V}_l$) such that $\exists k, i \in \mathcal{I}_{kl}$, the corresponding ranking vector is

$$\begin{aligned}\mathbf{r} &= \frac{1-c}{1+2a}(\mathbf{I} - \frac{c}{1+2a}\tilde{\mathbf{A}} - \frac{2a}{1+2a}\tilde{\mathbf{Y}})^{-1}\mathbf{e}_{l,i} \\ &= \frac{1-c}{1+2a}\sum_{p=0}^{\infty}(\frac{c}{1+2a}\tilde{\mathbf{A}} + \frac{2a}{1+2a}\tilde{\mathbf{Y}})^{p}\mathbf{e}_{l,i}\end{aligned} \tag{5.4}$$

where $\mathbf{e}_{l,i}(\sum_{q=1}^{l-1} n_q + i) = 1$ and 0 elsewhere, and n_q denotes the number of nodes in $\mathcal{G}_q$. We can see that the ranking score of node-j in $\mathcal{G}_k$ $\mathbf{r}(\sum_{q=1}^{k-1} n_q + j) > 0$ if and only if there exists a p such that $[\tilde{\mathbf{A}} + \tilde{\mathbf{Y}}]^p(\sum_{q=1}^{k-1} n_q + j, \sum_{q=1}^{l-1} n_q + i) > 0$, i.e., node-$j$ in $\mathcal{G}_k$ is reachable from node-i in $\mathcal{G}_l$ by p walks along both edges in the domain-specific networks and the edges in the main network. In this way, node-i can influence the nodes in the same domain as node-i and those in other domains. By horizontally concatenating the query vectors corresponding to all the common nodes, we achieve the following matrix-form ranking scores

$$\mathbf{R} = \frac{1-c}{1+2a}(\mathbf{I} - \frac{c}{1+2a}\tilde{\mathbf{A}} - \frac{2a}{1+2a}\tilde{\mathbf{Y}})^{-1}\mathbf{E} \tag{5.5}$$

where $\mathbf{R} \in \mathbb{R}^{n \times n_c}$, $n = \sum_{p=1}^{g} n_p$ denotes the total number of nodes in all domain-specific networks and $n_c = |\bigcup_{k,l=1}^{g} \mathcal{I}_{kl}|$ is the total number of common nodes across each two domain-specific networks. $\mathbf{E} \in \mathbb{R}^{n \times n_c}$ is the indicator matrix of the common nodes, e.g., $\mathbf{E}(\sum_{q=1}^{l-1} n_q + i, u) = 1$ if node-i in $\mathcal{G}_l$ is the u-th common nodes.

To adapt the ranking algorithm to learning node representations, we propose to feed Eq. (5.5) into a linear layer without bias, i.e.,

$$\mathbf{X} = \mathbf{RW} = \frac{1-c}{1+2a}(\mathbf{I} - \frac{c}{1+2a}\tilde{\mathbf{A}} - \frac{2a}{1+2a}\tilde{\mathbf{Y}})^{-1}\mathbf{W}_c \tag{5.6}$$

where $\mathbf{W}_c = \mathbf{EW} \in \mathbb{R}^{n \times d}$ is the parameter matrix and $\mathbf{W}_c(\sum_{q=1}^{l-1} n_q + i, :) = \mathbf{0}$ if node-i in the domain-specific network $\mathcal{G}_l$ is *not* shared with any other domains. In other words, the learnable parameters correspond to the initial embedding lookup table of the common nodes. Equivalently, we can understand $\mathbf{W}_c$ as the prediction step in [181] where we aim to 'predict' the representations of common nodes instead of predicting the node labels by input node features. The propagation step is done by $(\mathbf{I} - \frac{c}{1+2a}\tilde{\mathbf{A}} - \frac{2a}{1+2a}\tilde{\mathbf{Y}})^{-1}$, which requires an $O(n^3)$ computational complexity. To approximate the propagation step, as the eigenvalues of $\frac{c}{1+2a}\tilde{\mathbf{A}} + \frac{2a}{1+2a}\tilde{\mathbf{Y}}$ lie in $(-1, 1)$ [103], we apply a finite order of Taylor expansion as

$$\begin{aligned} \mathbf{X}_{(0)} &= \frac{1-c}{1+2a}\mathbf{W}_c \\ \mathbf{X}_{(p)} &= \frac{c}{1+2a}\tilde{\mathbf{A}}\mathbf{X}_{(p-1)} + \frac{2a}{1+2a}\tilde{\mathbf{Y}}\mathbf{X}_{(p-1)} + \mathbf{X}_{(0)} \\ \mathbf{X} &= \mathbf{X}_{(P)} \end{aligned} \tag{5.7}$$

whose time complexity is $O(m_A dP + m_Y dP)$ where m_A, m_Y denote the number of nonzero elements in $\tilde{\mathbf{A}}, \tilde{Y}$. The whole process is essentially to learn node representations by propagating the initial representations $\mathbf{W}_c$ of common nodes to their P-hop neighborhood.

5.3.3 DraNoN: Embedding on Dynamic NoN

When networks evolve over time, we are given a dynamic network of networks $\mathcal{G} = \{\mathcal{G}_0^1, \cdots, \mathcal{G}_0^T\}$ where $\mathcal{G}_0^t = \{\mathcal{V}_0^t, \mathbf{A}_0^t\}$ is the main network and $\mathcal{V}_0^t = \{\mathcal{G}_1^t, \cdots, \mathcal{G}_g^t\}$ represents domain-specific networks at snapshot t. We extend the static model by two components, each of which aims to capture different aspects of temporal evolution. In particular, we propose to use a GRU to capture the dynamics on the common nodes at different snapshots (i.e., $\mathbf{W}_c^t$) and a self-attention based module that learns how networks evolve by adaptively weighting the historical node representations.

Dynamics on Common Nodes. First, to model the dynamics behind the common nodes $\mathcal{I}_{kl}^t,\ k, l = 1, \cdots, g$, we consider the initial representations of these nodes $\mathbf{W}_c^t$ as a dynamical system and use GRU to encode both the current input node features $\mathbf{E}^t$ and the historical common node representations $\mathbf{W}_c^{t-1}$. By denoting $N_c = \bigcup_{t=1}^{T} \bigcup_{k,l=1}^{g} |\mathcal{I}_{kl}^t|$, i.e., the number of unique common nodes across all snapshots, we denote $\hat{\mathbf{W}}_c^t \in \mathbb{R}^{N_c \times d}$ as the representations of the common nodes at snapshot t such that $\mathbf{W}_c^t(\sum_{q=1}^{l-1} n_q + i, :) = \hat{\mathbf{W}}_c^t(u, :)$ if the node-i in $\mathcal{G}_l$ essentially corresponds to the u-th common node. Thus, we alternatively capture the dynamics by $\hat{\mathbf{W}}_c^t$ with GRU formulated as

$$\hat{\mathbf{W}}_c^t = \text{GRU}(\mathbf{E}^t, \hat{\mathbf{W}}_c^{t-1}). \tag{5.8}$$

where we expand $\mathbf{E}^t \in \mathbb{R}^{n \times n_c^t}$ to $\mathbf{E}^t \in \mathbb{R}^{n \times N_c}$ by inserting zero columns. The GRU cell is

$$\begin{aligned}
\mathbf{Z}^t &= \sigma(\mathbf{W}_Z \mathbf{H}^t + \mathbf{U}_Z \hat{\mathbf{W}}_c^{t-1} + \mathbf{B}_Z) \\
\mathbf{R}^t &= \sigma(\mathbf{W}_R \mathbf{H}^t + \mathbf{U}_R \hat{\mathbf{W}}_c^{t-1} + \mathbf{B}_R) \\
\bar{\mathbf{W}}_c^t &= \sigma_t(\mathbf{W}_H \mathbf{H}^t + \mathbf{U}_H(\mathbf{R}^t \odot \hat{\mathbf{W}}_c^{t-1}) + \mathbf{B}_H) \\
\hat{\mathbf{W}}_c^t &= (1 - \mathbf{Z}^t) \odot \hat{\mathbf{W}}_c^{t-1} + \mathbf{Z}^t \odot \bar{\mathbf{W}}_c^t
\end{aligned} \tag{5.9}$$

where $\mathbf{W}_Z, \mathbf{W}_R, \mathbf{W}_H, \mathbf{U}_Z, \mathbf{U}_R, \mathbf{U}_H \in \mathbb{R}^{N_c \times N_c}$ and $\mathbf{B}_Z, \mathbf{B}_R, \mathbf{B}_H \in \mathbb{R}^{N_c \times d}$. σ, σ_t represent the sigmoid and tangent activation functions. $\mathbf{H}^t$ is the summarization of the input node features $\mathbf{E}^t$, which we compute by self-attention based node importance [144]:

$$\begin{aligned}
\mathbf{z}^t &= (\frac{c}{1+2a}\tilde{\mathbf{A}} + \frac{2a}{1+2a}\tilde{\mathbf{Y}})\mathbf{E}^t \mathbf{w}_s \\
\mathcal{I}^t &= \text{top-indices}(\mathbf{z}^t, d) \\
\mathbf{H} &= [\mathbf{E}^t \odot \sigma_t(\mathbf{z}^t)](\mathcal{I}^t, :) \\
\mathbf{H}^t &= \mathbf{H}'
\end{aligned} \tag{5.10}$$

With the updated $\mathbf{W}_c^t$, the representations of nodes that appear at snapshot t are

$$\begin{aligned}\mathbf{X}_{(0)}^t &= \frac{1-c}{1+2a}\mathbf{W}_c^t \\ \mathbf{X}_{(p)}^t &= \frac{c}{1+2a}\tilde{\mathbf{A}}^t\mathbf{X}_{(p-1)}^t + \frac{2a}{1+2a}\tilde{\mathbf{Y}}^t\mathbf{X}_{(p-1)}^t + \mathbf{X}_{(0)}^t \\ \mathbf{X}_e^t &= \mathbf{X}_{(P)}^t\end{aligned} \tag{5.11}$$

However, by propagating the dynamic representations of common nodes to the rest of the nodes (i.e., Eq. (5.11), the dynamics behind them mainly reply on the dynamics of the common nodes. Thus, we need to additionally design a module that models the dynamics of nodes based on their own historical representations.

Dynamics based on Self-Attention. This component aims to capture the dynamics behind *all* the nodes in the domain-specific networks. In particular, given the success of attention based sequential learning, we consider a sequence of the representations of a certain node at different snapshots as the input to the temporal attention layer, similar to [75]. These input representations need to capture the structural information of nodes at different snapshots. In this work, we use Eq. (5.7) with a shared parameter matrix $\mathbf{W}_c$ across all snapshots to capture such structural information, i.e.,

$$\bar{\mathbf{X}}_{(p)}^t = \frac{c}{1+2a}\tilde{\mathbf{A}}^t\mathbf{X}_{(p-1)}^t + \frac{2a}{1+2a}\tilde{\mathbf{Y}}^t\mathbf{X}_{(p-1)}^t + \mathbf{X}_{(0)}^t \tag{5.12}$$

where $\mathbf{X}_{(0)}^t = \frac{1-c}{1+2a}\mathbf{W}_c,\ \forall t = 1, \cdots, T$ and the output representations at snapshot t are denoted by $\bar{\mathbf{X}}^t = \mathbf{X}_{(P)}^t$. Then the sequence of representations for node-i in $\mathcal{G}_l$ is packed into an embedding matrix $\bar{\mathbf{Y}}_{l,i} \in \mathbb{R}^{T\times d}$ where $\bar{\mathbf{Y}}_{l,i}(t,:) = \bar{\mathbf{X}}^t(\sum_{p=1}^{l-1} n_p + i, :)$.

This temporal attention layer aims to learn the weights that aggregate the historical node representations. These weights are learned by the scaled dot-product form of attention [75]. In this way, the temporal self-attention layer is

$$\mathbf{Y}_{l,i} = \boldsymbol{\beta}_{l,i}(\bar{\mathbf{Y}}_{l,i}\mathbf{W}_v) \tag{5.13}$$

where the attention weights $\boldsymbol{\beta}_{l,i} \in \mathbb{R}^{T\times T}$ can be computed by

$$\begin{aligned}\boldsymbol{\beta}_{l,i}(t_1,t_2) &= \frac{\exp(\boldsymbol{\alpha}_{l,i}(t_1,t_2))}{\sum_{t_3=1}^{T}\exp(\boldsymbol{\alpha}_{l,i}(t_1,t_3))} \\ \boldsymbol{\alpha}_{l,i}(t_1,t_2) &= \frac{\left(\bar{\mathbf{Y}}_{l,i}\mathbf{W}_q)(\bar{\mathbf{Y}}_{l,i}\mathbf{W}_k)'\right)(t_1,t_2)}{\sqrt{d}} + \mathbf{M}(t_1,t_2)\end{aligned} \tag{5.14}$$

Here $\mathbf{M}$ is a mask matrix that ensures only representations from past snapshots contribute to the current snapshot. It is set as $\mathbf{M}(t_1, t_2) = 0$ if $t_1 \leq t_2$ and $\mathbf{M}(t_1, t_2) = -\infty$ otherwise. Then we construct the output node representations $\mathbf{X}_s^t$ of this self-attention based temporal layer by $\mathbf{X}_s^t(\sum_{p=1}^{l-1} n_p + i, :) = \mathbf{Y}_{l,i}(t, :),\ \forall t = 1, \cdots, T$.

Entire Dynamic Model. In order to compute final output node representations, we apply a linear layer upon the concatenation of $\mathbf{X}_e^t$ and $\mathbf{X}_s^t$ at snapshot t, i.e.,

$$\mathbf{X}^t = [(\mathbf{X}_e^t)' \| (\mathbf{X}_s^t)']' \mathbf{W} + \mathbf{b} \tag{5.15}$$

where $\mathbf{W} \in \mathbb{R}^{2d \times d}$ denotes the parameter matrix of the linear layer.

5.3.4 Model Training

In this work, we apply the proposed model DraNoN to the task of dynamic network alignment. The dynamic network alignment problem can be reformulated as the problem in the context of dynamic NoN as following. At each snapshot t, we are given a set of known common nodes (i.e., anchor nodes that are known a priori), and we aim to predict the rest of common nodes (i.e., unknown node alignments across each two domain-specific networks). Specifically, given a set of anchor nodes $\mathcal{I}_{kl}^t,\ \forall k, l = 1, \cdots, g$ between networks $\mathcal{G}_k^t$ and $\mathcal{G}_l^t$, we minimize the following binary cross-entropy loss

$$\begin{aligned} J = &- \sum_{t=1}^{T} \sum_{l=2}^{g} \sum_{k=1}^{l-1} \sum_{u \in \mathcal{I}_{kl}^t} 2 \log(\sigma(f(\mathbf{x}_{l,i}^t, \mathbf{x}_{k,j}^t))) \\ &+ \sum_{v=1}^{N} \log(-\sigma(f(\mathbf{x}_{l,i}^t, \mathbf{x}_{k,j_v}^t))) + \log(-\sigma(f(\mathbf{x}_{l,i_v}^t, \mathbf{x}_{k,j}^t))) \end{aligned} \tag{5.16}$$

where node-i in $\mathcal{G}_l^t$ and node-j in $\mathcal{G}_k^t$ represent the same entity u. In addition, node-i_v in $\mathcal{G}_l^t$ and node-j_v in $\mathcal{G}_k^t$ are the cross-network negative samples for anchor node-j in $\mathcal{G}_k^t$ and anchor node-i in $\mathcal{G}_l^t$, respectively and N denotes the number of negative samples. The scoring function f is defined as $f(\mathbf{x}_{l,i}^t, \mathbf{x}_{k,j}^t) = -\|\mathbf{x}_{l,i}^t - \mathbf{x}_{k,j}^t\|_1$. For an anchor node, the negative samples are generated by selecting nodes with the top-N scores excluding the anchor node itself [32].

Implementation details. To further improve the model capability of aligning nodes at different snapshots, we propose to use pseudo labelling to augment labels (i.e., anchor links) at snapshot t based on the node pairs that are aligned with highest confidence at previous snapshots. In practice, we select 10% node pairs that have highest scores defined by $f(\cdot, \cdot)$ as the pseudo labels for future snapshots. Moreover, since the computations of attention

coefficients (Eq. (5.14)) could be memory consuming, we carefully implement the temporal attention layer by using sparse matrices to improve the efficiency.

5.4 EXPERIMENTAL EVALUATIONS

We evaluate the proposed DraNoN in the task of dynamic network alignment. Specifically, we evaluate it in the following aspects:

- Q1: How accurate is it for dynamic network alignment?
- Q2: To what extent does the proposed method benefit from different components of the model?

5.4.1 Experimental Setup

Datasets. We evaluate dynamic network alignment in three scenarios whose dataset statistics are summarized in Table 5.2. Note that due to the difficulties in exploring data, we consider pairwise network alignment (i.e., $g = 2$) and we set $\mathbf{A}_0(1, 2) = 1$. The datasets that we use in the experiments include:

- *ACM co-author networks.* The ACM citation dataset was collected in 2017 including 2,385,022 papers with the author and venue information of each paper [150]. We extract papers published in the areas of data mining, database and information retrieval during the years 2000-2013. We construct the dynamic co-author networks by using the collaborations among authors who published papers in a certain year as the edges at the corresponding snapshot. Accordingly, the nodes of the networks represent the authors.
- *DBLP co-author networks.* The DBLP citation dataset was collected in 2017 including 3,680,007 papers with the author and venue information of each paper [150]. We construct the dynamic DBLP co-author networks in a similar way to the ACM co-author networks.
- *UCI communication networks.* This dataset [182] includes the messages sent among students in University of California, Irvine on its online social network platform. The edges of the networks indicate that one student sends a message to the other. Each edge is associated with a timestamp, based on which we slide the entire dataset into 13 snapshots [75].

Table 5.2: Data statistics.

	ACM	DBLP	UCI	UCI-N	AS	AS-N
# of nodes	17,695	17,022	1,809	1,809	6,506	6,506
# of edges	45,920	44,080	13,228	20,807	13,466	27,375
# of snapshots	14	14	13	13	20	20
# of alignments	13,726		1,796		6,394	

- *Autonomous system networks.* The dataset contains 733 daily instances which range from November 8th, 1997 to January 2nd, 2000 [183]. Nodes of the networks represent autonomous systems and there exists an edge in the networks if one autonomous system (AS) exchanges traffic flows with the other. In the experiments, we select 20 of all instances as the snapshot networks.

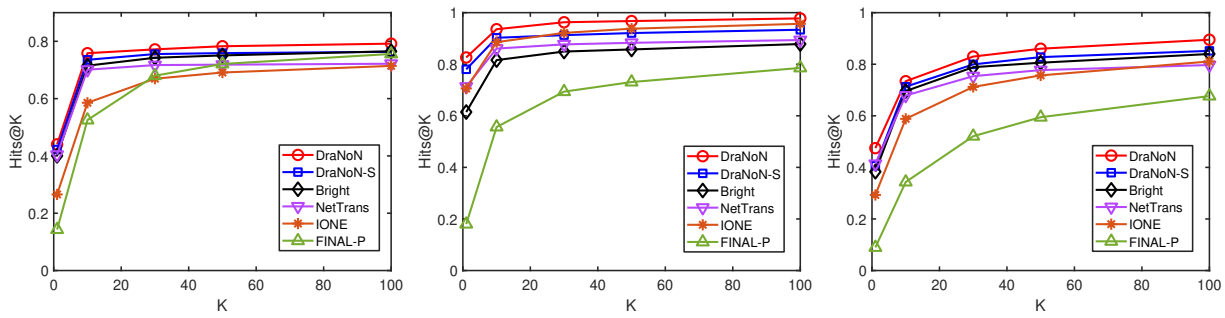
With the above datasets, we build three alignment scenarios:

- *ACM vs.DBLP.* The ACM co-author networks and DBLP co-author networks share 13,726 nodes (i.e., ground-truth node alignments). We randomly select 20% common nodes (i.e., anchor nodes) as the training data and test on the rest of 11,582 alignment pairs. Note that at each snapshot, only part of these alignments are available.

- *UCI vs. UCI-N.* Given the dynamic UCI communication networks, we generate their noisy permuted counterpart by randomly rewiring edges and changing the timestamps associated with edges. The detailed process can be referred to [67]. Note that this process could lead to node deletions in the noisy counterparts. There exist 1,796 ground-truth alignments. We use 20% of them as the training data.

- *AS vs. AS-N.* It is constructed similarly as the above scenario.

Baseline Methods. We compare our proposed DraNoN method with the following static network alignment methods, including: (1) FINAL-P [9], (2) IONE [8], (3) NetTrans [14], Bright [32], and (4) DraNoN-S, the static variant of our proposed model.
Evaluation Metrics. We evaluate the effectiveness by Hits@K. Given the test node alignments (e.g., node-i in $\mathcal{G}_1$ and node-j in $\mathcal{G}_2$), if $f(\mathbf{x}_{1,i}, \mathbf{x}_{2,j})$ is among the highest top-K values within the nodes in $\mathcal{G}_2$, then we say there is a *hit*. We count the number of hits, divided by the total number of test node alignment.
Machine. Our proposed model is implemented by PyTorch. We use Nvidia V100 with 32G memory as GPU to run the graph neural network based methods. We set $a = 1.0, c = 0.85, P = 10$ and the dimension of node representations as $d = 128$. The hyperparameters of the baseline methods are set as default.



(a) ACM vs. DBLP networks. (b) UCI vs. UCI-N networks. (c) AS vs. AS-N networks.

Figure 5.4: Results of aligning among all nodes.

5.4.2 Dynamic Network Alignment

We compare our proposed method DraNoN with baseline methods from two perspectives. First, we want to measure how effective the methods are in aligning networks at each snapshot. That is, at each time snapshot t, we find node alignments between $\mathcal{V}_1^t$ and $\mathcal{V}_2^t$. The results are shown in Figure 5.6. We have the following observations. First, our proposed method DraNoN outperforms all the baseline methods, which indicates that the dynamic information can indeed assist the alignment at each snapshot. Specifically, compared with the best competitor (e.g., NetTrans [14] and Bright [32]), our proposed method achieves an up to 5.6%, 14% and 2.5% improvement in Hits@30 on the three alignment scenarios, respectively. Second, the comparisons between DraNoN and DraNoN-S also demonstrate the necessity of the dynamic module for more accurate alignment inference on dynamic networks. Third, even at the early snapshots (e.g., $t = 1$), we can still observe the improvements of DraNoN over other static methods, which could be because learning GRU over the entire time span advances in learning the parameter matrix $\mathbf{W}_c^1$ and hence better node representations for alignment inference. Last, our proposed static model variant DraNoN-S can achieve a better alignment performance in most alignment scenarios, which validates its effectiveness in the static setting.

Furthermore, we also want to measure the performance of aligning nodes among the entire set of nodes. That is, we align nodes between the entire node sets of two networks throughout the time span, i.e., aligning $\bigcup_{t=1}^{T} \mathcal{V}_1^t$ with $\bigcup_{t=1}^{T} \mathcal{V}_2^t$. Here, the static alignment methods are conducted on the static networks by merging all snapshot networks. The results are summarized in Figure 5.4. As one can see, our proposed method DraNoN achieves a better performance of Hits@K for $K = 1, 10, 30, 50, 100$ than other baseline methods. For instance, our proposed method improves the state-of-the-arts by at least 3% in Hits@30. From this perspective, this also demonstrates the importance of leveraging the temporal

information behind dynamic networks.

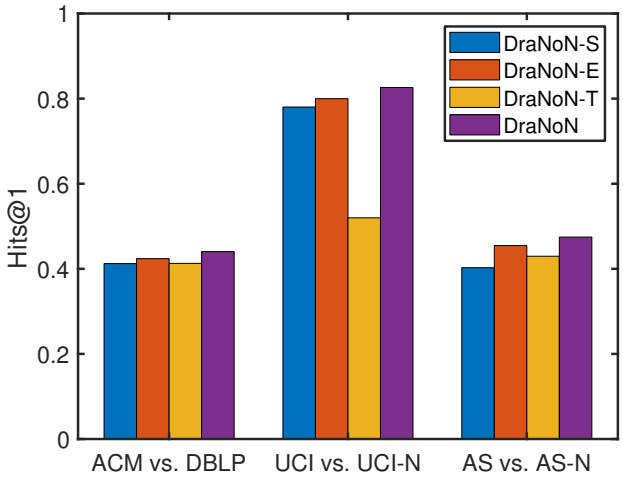
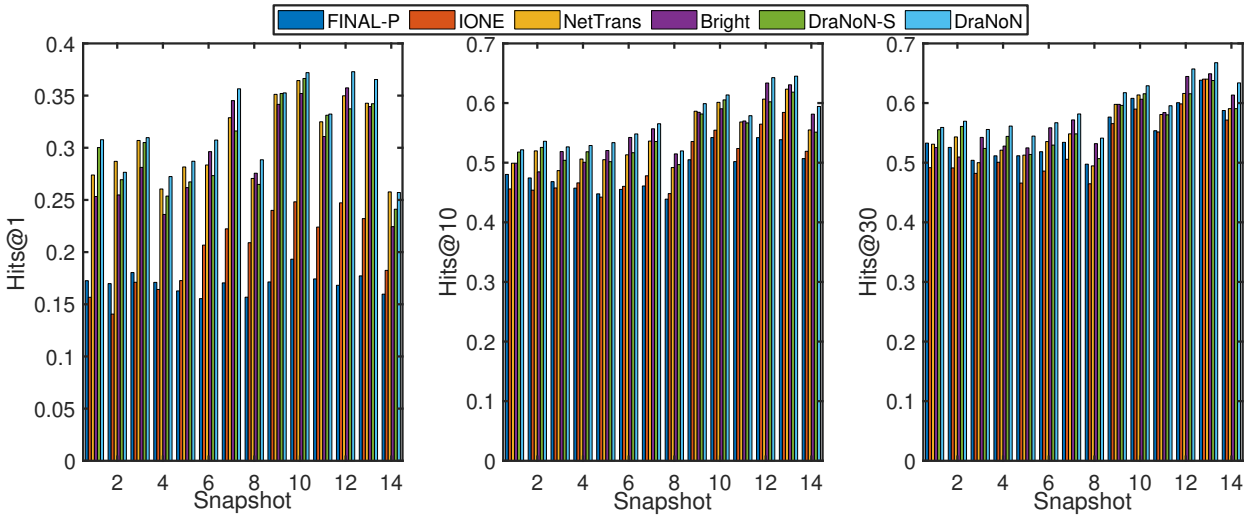
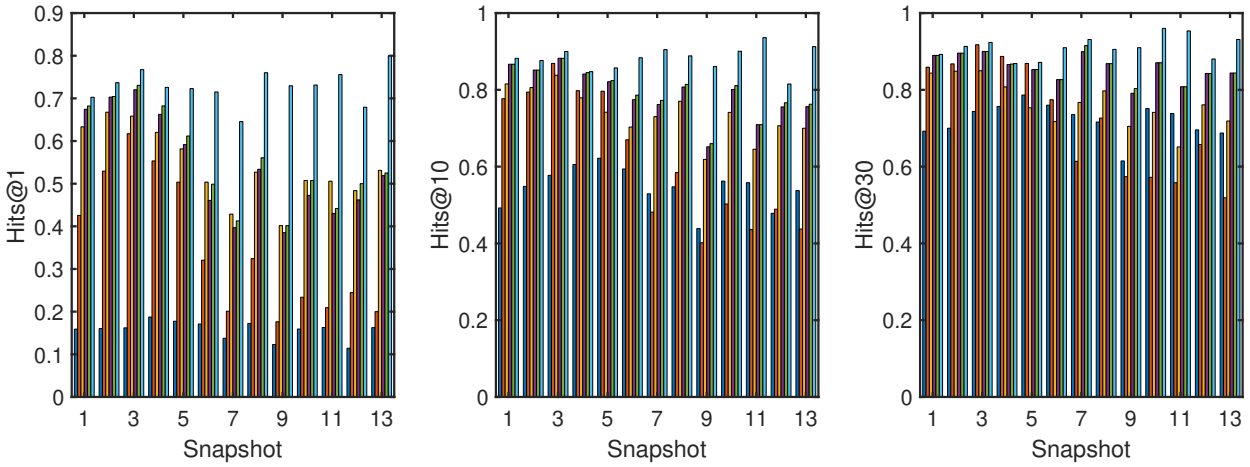


Figure 5.5: Ablation study on model design.

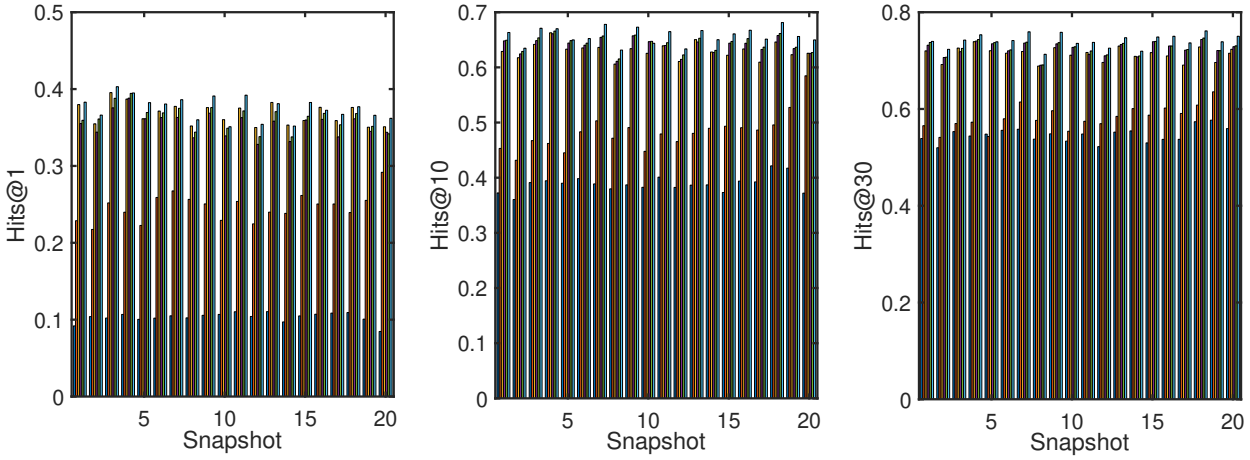
Ablation study on model design. We conduct ablation studies on different model variants of our proposed DraNoN. Specifically, we compare the full model DraNoN with the following variants, including: (1) DraNoN-S which is the static variant without using any temporal information, (2) DraNoN-E which merely uses the GRU to capture the dynamics behind common nodes, and (3) DraNoN-T which only applies the temporal attention layer. We evaluate the performance by Hits@1 when aligning $\bigcup_{t=1}^{T} \mathcal{V}_1^t$ with $\bigcup_{t=1}^{T} \mathcal{V}_2^t$. The results are shown in Figure 5.5. As we can observe, DraNoN-E achieves a better performance than the static variant DraNoN-S, demonstrating the effectiveness of capturing the dynamics behind the known common nodes (i.e., anchor nodes). In addition, although DraNoN-T may slightly improve the performance on certain datasets, it sometimes hurts the alignment performance. Last, the full model DraNoN consistently outperforms all the other model variants, validating the necessity of all the components.



(a) Results on ACM vs. DBLP co-author networks.



(b) Results on UCI vs. UCI-N communication networks.



(c) Results on AS vs. AS-N networks.

Figure 5.6: Results of aligning different snapshot networks.

CHAPTER 6: CONCLUSION AND FUTURE DIRECTIONS

In this chapter, we summarize our key research contributions and discuss the future research directions in network alignment.

6.1 CONCLUSION

In this thesis, we study the network alignment problem on big networks and develop several works to address the challenges associated with the 4Vs characteristics of big data, including variety, veracity, velocity and volume. Note that we consider volume as a fundamental factor such that the designed methods have an at least comparable efficiency with prior works.

Task 1 - Variety in Network Alignment. *Attributes of Networks.* To leverage the attribute information, we develop an attributed network alignment method, whose key idea is formulate the alignment consistency principle that includes topological consistency and attribute consistency into an optimization problem. We develop a family of iterative fixed-point algorithms named FINAL to efficiently solve this optimization problem. We theoretically prove that the FINAL algorithms converge to the global optimal solution. The experiments demonstrate the algorithms FINAL achieve an up to 30% alignment accuracy improvement compared with the structure-only based alignment methods and their variants using attributes heuristically. In addition, we also leverage low-rank approximation techniques and multiresolution matrix factorization to further scale up FINAL.
Disparity behind Networks. To mitigate the disparity issues among different networks, we first develop a semi-supervised alignment method named ORIGIN to address the embedding space disparity issue of different networks. Specifically, ORIGIN learns a non-rigid displacement function that can move the node embeddings of one network towards the embeddings of another network. We conduct experiments that validate the better performance of ORIGIN than the baseline methods. We also visualize the node embeddings of two networks before and after applying the non-rigid displacement function, which visually demonstrates the space disparity issue can be mitigated.

Second, we design an end-to-end model NetTrans to solve the cross-network transformation problem. We do not explicitly assume the alignment consistency principle but instead use the learned cross-network transformation functions to encode the network disparity. The key idea is to coarsen the input source network via pooling layers and hierarchically reconstruct the target network by unpooling layers. We conduct extensive experiments on

network alignment, which show that NetTrans outperforms other state-of-the-arts consistency based methods and embedding based methods by up to 6.5% in terms of Hits@30.

Third, we also study how to strike a balance between alignment consistency and alignment disparity. In particular, we design a relational graph convolutional layer whose output node embeddings can infer node alignments while preserving alignment consistency in a similar way as FINAL. We then develop a novel negative sampling strategy to impose alignment disparity. By integrating these components, the developed algorithm NeXtAlign achieves significant improvements compared with other state-of-the-arts methods.

Task 2 - Veracity in Network Alignment. To handle the incompleteness of networks, we propose to jointly solve network alignment and network completion problems such that they can mutually benefit each other. To efficiently compute the alignment matrix, we theoretically uncover the low-rank structure of the alignment matrix. Based on the low-rank characteristic of networks, we develop a multiplicative update algorithm iNeatẆe conduct extensive experiments which show that iNeat achieves an up to 30% improvement in alignment accuracy and a higher missing edge recovery rate than the baseline methods. Moreover, we also theoretically study how robust the proposed FINAL algorithms are against the structural perturbations. Specifically, we present the error bound between the alignment matrix given the perturbed adjacency matrices and that given the original networks. The key idea is to consider the closed-form solution of FINAL as a linear system and hence utilize the stability analysis of linear systems.

Task 3 - Velocity in Network Alignment. For this task, we study the dynamic network alignment problem where the input networks dynamically change over time. To be more specific, the goal is to leverage the temporal information to boost the alignment performance. We first model multiple dynamic networks to be aligned into dynamic network of networks, and then the dynamic network alignment problem can be reformulated as a common node prediction problem in dynamic network of networks. To solve this problem, we design a graph neural network model DraNoN composed of a GRU and a temporal attention layer to learn dynamic node representations. Experiments demonstrate that DraNoN achieves a much better performance than static alignment methods.

6.2 FUTURE DIRECTIONS

Despite the extensive research, network alignment still remains an active area of explorations. We present several promising research directions as follows.

Seed Selection in Network Alignment. The goal of this direction is to select a set of node pairs that can boost the performance of network alignment. It is of great necessity in unsupervised network alignment and active network alignment problems.

In the unsupervised setting, many existing alignment methods assimilate the prior node similarity matrix into the algorithm. For example, network alignment methods often computes the prior similarity matrix by BLAST scores [3], graphlet-based similarity scores [184], degree-based similarities [9], etc. These similarity matrices can be considered as a *soft* seed matrix (i.e., prior many-to-many alignments) used in the alignment algorithm. Thus, whether or not good (*soft*) seed alignment pairs can be selected could largely determine the performance of the entire unsupervised alignment algorithms. In addition to exploring external similarity matrix which often requires domain knowledge, one possible direction is to utilize pre-training techniques to learn node embeddings. By measuring the similarities among these embeddings, a set of (*soft*) seed alignments can be selected. Yet, it is challenging to design the pre-training model that can learn comparable node embeddings across different networks.

Another setting is when a human annotator could tell a correct alignment given some query node and this new alignment pair will be used as the anchor link (i.e., seed) to aid the subsequent learning process. This is known as active network alignment. Existing methods use distribution based method and influence function based method to select query nodes for the human annotator. However, given the complex patterns underlying multiple networks, how to effectively design active learning method for network alignment is an interesting yet nontrivial direction to discover.

Adversarial Network Alignment. Adversarial graph learning has been recently studied and applied to many downstream tasks, including node classification, graph classification, link prediction, fraud detection and so on. Nevertheless, few works apply adversarial learning to network alignment. In general, adversarial learning can be divided into two directions: adversarial attacks and adversarial defense. The goal of adversarial attacks on network alignment is to attack the structure and/or attributes of one or multiple networks such that the performance of the alignment algorithms decreases. Existing works mainly focus on attacking a single network. However, attacks on multiple networks could become more complicated as one attack on a network might influence both the network itself and other networks. From this perspective, adversarial attack on network alignment is still an under-explored research topic. In the meanwhile, adversarial defense on network alignment, which aims to develop resilient techniques to improve the robustness of the model against adversarial attacks, is another meaningful research problem to study.

Online Network Alignment. Since networks often change over time and existing network alignment methods are often costly, it is of great interests to efficiently update the alignment matrix once the input network structure changes. For example, given two social networks (e.g., Facebook and Twitter), suppose the alignment matrix at time $t = t_0$ is already obtained, then how can we compute the alignment matrix at time $t = t_1, t_2, \cdots$? The most straightforward way is to calculate the alignment matrix by some existing approach from scratch, which however is computationally expensive and unnecessary. In this way, how to efficiently update the alignment matrix (e.g., by updating only a certain portion of the matrix) is still an open question.

Integrated Network Alignment. Network alignment can be integrated with many other classic machine learning tasks, such as fairness, interpretability, etc. However, this direction is still overlooked. First, many real-world networks follow a power-law degree distribution such that the majority of nodes have low degrees. These low-degree nodes could be a group of underrepresented users in social networks. However, in the task of network alignment, the alignments on low-degree nodes are often misleading possibly due to the fact that low-degree nodes have similar and somewhat indistinguishable structural patterns. Despite some existing works on fair graph learning [185, 186, 187], they focus on tasks on a single network, while leaving fairness on multiple networks blank. In this way, by studying fair network alignment, both the fairness and alignment accuracy can be improved.

Second, most, if not all, of the existing network alignment methods seek for a better alignment performance, while not attempting to explain why nodes are aligned. The interpretibility is an important factor that should be valued especially in many industrial applications. For example, by interpreting why entities are aligned across two knowledge graphs, one can evaluate the confidence of two nodes being the same/similar entities (e.g., objects, concepts, etc.). In the application of security (e.g., fraud detection, modeling adversarial activities, etc.), it is indeed important provide clues of identifying the fraudsters across multi-sourced networks, but explaining why they are aligned could be even more crucial for security department to take actions. As a result, interpretable network alignment is another interesting research direction.

Beyond Network Alignment. In contrast to network alignment which aims to merge networks into a world-view network based on the common nodes across networks, a reverse direction is to split a network into multiple pieces. Instances of network splitting include graph partitioning, clustering, subgraph pattern mining, community detection, etc. To be more specific, overlapped community detection divides a network into multiple communities

with overlapped nodes. From this perspective, overlapped community detection also shares the commonality with network alignment, i.e., both explicitly or implicitly identifying the shared nodes across different pieces/networks. Given these differences and commonalities between network alignment and network splitting, a natural research question is when to align networks and when to split networks. In addition, a future direction is to think about whether there exists any room to utilize both network merging and network splitting to benefit certain applications.

REFERENCES

[1] Q. Zhan, J. Zhang, S. Wang, S. Y. Philip, and J. Xie, "Influence maximization across partially aligned heterogenous social networks," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2015, pp. 58–69.

[2] F. E. Faisal, L. Meng, J. Crawford, and T. Milenković, "The post-genomic era of biological network alignment," *EURASIP Journal on Bioinformatics and Systems Biology*, vol. 2015, no. 1, pp. 1–19, 2015.

[3] 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.

[4] J. Xu, H. Tong, T.-C. Lu, J. He, and N. Bliss, "Gta3 2018: Workshop on graph techniques for adversarial activity analytics," in *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, 2018, pp. 803–803.

[5] M. Bayati, M. Gerritsen, D. F. Gleich, A. Saberi, and Y. Wang, "Algorithms for large, sparse network alignment problems," in *2009 Ninth IEEE International Conference on Data Mining*. IEEE, 2009, pp. 705–710.

[6] J. Zhang and S. Y. Philip, "Multiple anonymized social networks alignment," in *2015 IEEE International Conference on Data Mining*. IEEE, 2015, pp. 599–608.

[7] 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, pp. 1485–1494.

[8] L. Liu, W. K. Cheung, X. Li, and L. Liao, "Aligning users across social networks using network embedding." in *Ijcai*, 2016, pp. 1774–1780.

[9] 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.

[10] S. Zhang, H. Tong, J. Tang, J. Xu, and W. Fan, "ineat: Incomplete network alignment," in *2017 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2017, pp. 1189–1194.

[11] S. Zhang, H. Tong, R. Maciejewski, and T. Eliassi-Rad, "Multilevel network alignment," in *The World Wide Web Conference*, 2019, pp. 2344–2354.

[12] S. Zhang and H. Tong, "Attributed network alignment: Problem definitions and fast solutions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 9, pp. 1680–1692, 2018.

[13] S. Zhang, H. Tong, J. Xu, Y. Hu, and R. Maciejewski, "Origin: Non-rigid network alignment," in *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019, pp. 998–1007.

[14] S. Zhang, H. Tong, Y. Xia, L. Xiong, and J. Xu, "Nettrans: Neural cross-network transformation," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 986–996.

[15] S. Zhang, H. Tong, L. Jin, Y. Xia, and Y. Guo, "Balancing consistency and disparity in network alignment," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 2212–2222.

[16] S. Zhang, H. Tong, J. Tang, J. Xu, and W. Fan, "Incomplete network alignment: Problem definitions and fast solutions," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 14, no. 4, pp. 1–26, 2020.

[17] 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.

[18] 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.

[19] V. Saraph and T. Milenković, "Magna: maximizing accuracy in global network alignment," *Bioinformatics*, vol. 30, no. 20, pp. 2931–2940, 2014.

[20] V. Vijayan, V. Saraph, and T. Milenković, "Magna++: maximizing accuracy in global network alignment via both node and edge conservation," *Bioinformatics*, vol. 31, no. 14, pp. 2409–2411, 2015.

[21] S. Umeyama, "An eigendecomposition approach to weighted graph matching problems," *IEEE transactions on pattern analysis and machine intelligence*, vol. 10, no. 5, pp. 695–703, 1988.

[22] B. Luo and E. R. Hancock, "Structural graph matching using the em algorithm and singular value decomposition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 10, pp. 1120–1136, 2001.

[23] C. Ding, T. Li, and M. I. Jordan, "Nonnegative matrix factorization for combinatorial optimization: Spectral clustering, graph matching, and clique finding," in *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 2008, pp. 183–192.

[24] T. Man, H. Shen, S. Liu, X. Jin, and X. Cheng, "Predict anchor links across social networks via an embedding approach." in *Ijcai*, vol. 16, 2016, pp. 1823–1829.

[25] F. Zhou, L. Liu, K. Zhang, G. Trajcevski, J. Wu, and T. Zhong, "Deeplink: A deep learning approach for user identity linkage," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 1313–1321.

[26] H. Xu, D. Luo, H. Zha, and L. C. Duke, "Gromov-wasserstein learning for graph matching and node embedding," in *International conference on machine learning*. PMLR, 2019, pp. 6932–6941.

[27] 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.

[28] X. Chen, M. Heimann, F. Vahedian, and D. Koutra, "Cone-align: Consistent network alignment with proximity-preserving node embedding," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 1985–1988.

[29] 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.

[30] J. Zhang, B. Chen, X. Wang, H. Chen, C. Li, F. Jin, G. Song, and Y. Zhang, "Mego2vec: Embedding matched ego networks for user alignment across social networks," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, 2018, pp. 327–336.

[31] 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.

[32] Y. Yan, S. Zhang, and H. Tong, "Bright: A bridging algorithm for network alignment," in *Proceedings of the Web Conference 2021*, 2021, pp. 3907–3917.

[33] F. Zhou and F. De la Torre, "Factorized graph matching," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2012, pp. 127–134.

[34] F. Zhou and F. De la Torre, "Deformable graph matching," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 2922–2929.

[35] A. Zanfir and C. Sminchisescu, "Deep learning of graph matching," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2684–2693.

[36] M. Kim and J. Leskovec, "The network completion problem: Inferring missing nodes and edges in networks," in *Proceedings of the 2011 SIAM International Conference on Data Mining*. SIAM, 2011, pp. 47–58.

[37] F. Masrour, I. Barjesteh, R. Forsati, A.-H. Esfahanian, and H. Radha, "Network completion with node similarity: A matrix completion approach with provable guarantees," in *2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 2015, pp. 302–307.

[38] D. Rafailidis and F. Crestani, "Network completion via joint node clustering and similarity learning," in *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 2016, pp. 63–68.

[39] S. Soundarajan, T. Eliassi-Rad, B. Gallagher, and A. Pinar, "Maxreach: Reducing network incompleteness through node probes," in *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 2016, pp. 152–157.

[40] N. Boumal and P.-a. Absil, "Rtrmc: A riemannian trust-region method for low-rank matrix completion," in *Advances in neural information processing systems*, 2011, pp. 406–414.

[41] J.-F. Cai, E. J. Candès, and Z. Shen, "A singular value thresholding algorithm for matrix completion," *SIAM Journal on optimization*, vol. 20, no. 4, pp. 1956–1982, 2010.

[42] K.-C. Toh and S. Yun, "An accelerated proximal gradient algorithm for nuclear norm regularized linear least squares problems," *Pacific Journal of optimization*, vol. 6, no. 615-640, p. 15, 2010.

[43] E. J. Candes and Y. Plan, "Matrix completion with noise," *Proceedings of the IEEE*, vol. 98, no. 6, pp. 925–936, 2010.

[44] M. Al Hasan, V. Chaoji, S. Salem, and M. Zaki, "Link prediction using supervised learning," in *SDM06: workshop on link analysis, counter-terrorism and security*, vol. 30, 2006, pp. 798–805.

[45] W. Cukierski, B. Hamner, and B. Yang, "Graph-based features for supervised link prediction," in *The 2011 International Joint Conference on Neural Networks*. IEEE, 2011, pp. 1237–1244.

[46] A. K. Menon and C. Elkan, "Link prediction via matrix factorization," in *Joint european conference on machine learning and knowledge discovery in databases*. Springer, 2011, pp. 437–452.

[47] R. Raymond and H. Kashima, "Fast and scalable algorithms for semi-supervised link prediction on static and dynamic graphs," in *Joint european conference on machine learning and knowledge discovery in databases*. Springer, 2010, pp. 131–147.

[48] H. Kashima, T. Kato, Y. Yamanishi, M. Sugiyama, and K. Tsuda, "Link propagation: A fast semi-supervised learning algorithm for link prediction," in *Proceedings of the 2009 SIAM international conference on data mining*. SIAM, 2009, pp. 1100–1111.

[49] T.-T. Kuo, R. Yan, Y.-Y. Huang, P.-H. Kung, and S.-D. Lin, "Unsupervised link prediction using aggregative statistics on heterogeneous social networks," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013, pp. 775–783.

[50] R.-H. Li, J. X. Yu, and J. Liu, "Link prediction: the power of maximal entropy random walk," in *Proceedings of the 20th ACM international conference on Information and knowledge management*, 2011, pp. 1147–1156.

[51] X. Du, J. Yan, and H. Zha, "Joint link prediction and network alignment via cross-graph embedding." in *IJCAI*, 2019, pp. 2251–2257.

[52] L. H. Phuc, K. Takeuchi, M. Yamada, and H. Kashima, "Simultaneous link prediction on unaligned networks using graph embedding and optimal transport," in *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 2020, pp. 245–254.

[53] J. Zhang, J. Chen, S. Zhi, Y. Chang, S. Y. Philip, and J. Han, "Link prediction across aligned networks with sparse and low rank matrix estimation," in *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, 2017, pp. 971–982.

[54] F. Zhou, C. Li, Z. Wen, T. Zhong, G. Trajcevski, and A. Khokhar, "Uncertainty-aware network alignment," *International Journal of Intelligent Systems*, 2021.

[55] A. Bojchevski and S. Günnemann, "Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking," *arXiv preprint arXiv:1707.03815*, 2017.

[56] D. Zhu, P. Cui, D. Wang, and W. Zhu, "Deep variational network embedding in wasserstein space," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2827–2836.

[57] Z. Kang, H. Pan, S. C. Hoi, and Z. Xu, "Robust graph learning from noisy data," *IEEE transactions on cybernetics*, vol. 50, no. 5, pp. 1833–1843, 2019.

[58] J. Wang, Z. Li, Q. Long, W. Zhang, G. Song, and C. Shi, "Learning node representations from noisy graph structures," in *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2020, pp. 1310–1315.

[59] Q. Zhou, L. Li, N. Cao, L. Ying, and H. Tong, "Admiring: Adversarial multi-network mining," in *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2019, pp. 1522–1527.

[60] Z. Zhang, Z. Zhang, Y. Zhou, Y. Shen, R. Jin, and D. Dou, "Adversarial attacks on deep graph matching," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[61] Y. Zhou, Z. Zhang, S. Wu, V. Sheng, X. Han, Z. Zhang, and R. Jin, "Robust network alignment via attack signal scaling and adversarial perturbation elimination," in *Proceedings of the Web Conference 2021*, 2021, pp. 3884–3895.

[62] J. Ren, Z. Zhang, J. Jin, X. Zhao, S. Wu, Y. Zhou, Y. Shen, T. Che, R. Jin, and D. Dou, "Integrated defense for resilient graph matching," in *International Conference on Machine Learning*. PMLR, 2021, pp. 8982–8997.

[63] J. Ren, Y. Zhou, R. Jin, Z. Zhang, D. Dou, and P. Wang, "Dual adversarial learning based network alignment," in *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2019, pp. 1288–1293.

[64] H. Hong, X. Li, Y. Pan, and I. Tsang, "Domain-adversarial network alignment," *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[65] J. Gao, X. Huang, and J. Li, "Unsupervised graph alignment with wasserstein distance discriminator," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 426–435.

[66] V. Vijayan, D. Critchlow, and T. Milenković, "Alignment of dynamic networks," *Bioinformatics*, vol. 33, no. 14, pp. i180–i189, 2017.

[67] D. Aparício, P. Ribeiro, T. Milenković, and F. Silva, "Got-wave: Temporal network alignment using graphlet-orbit transitions," *arXiv preprint arXiv:1808.08195*, 2018.

[68] Y. Sun, J. Crawford, J. Tang, and T. Milenković, "Simultaneous optimization of both node and edge conservation in network alignment via wave," in *International Workshop on Algorithms in Bioinformatics*. Springer, 2015, pp. 16–39.

[69] L. Sun, Z. Zhang, P. Ji, J. Wen, S. Su, and S. Y. Philip, "Dna: Dynamic social network alignment," in *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019, pp. 1224–1231.

[70] L.-k. Zhou, Y. Yang, X. Ren, F. Wu, and Y. Zhuang, "Dynamic network embedding by modeling triadic closure process." in *AAAI*, 2018, pp. 571–578.

[71] L. Du, Y. Wang, G. Song, Z. Lu, and J. Wang, "Dynamic network embedding: An extended approach for skip-gram based network embedding." in *IJCAI*, 2018, pp. 2086–2092.

[72] P. Goyal, N. Kamra, X. He, and Y. Liu, "Dyngem: Deep embedding method for dynamic graphs," *arXiv preprint arXiv:1805.11273*, 2018.

[73] A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, T. B. Schardl, and C. E. Leiserson, "Evolvegcn: Evolving graph convolutional networks for dynamic graphs." in *AAAI*, 2020, pp. 5363–5370.

[74] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson, "Structured sequence modeling with graph convolutional recurrent networks," in *International Conference on Neural Information Processing*. Springer, 2018, pp. 362–373.

[75] A. Sankar, Y. Wu, L. Gou, W. Zhang, and H. Yang, "Dysat: Deep neural representation learning on dynamic graphs via self-attention networks," in *Proceedings of the 13th International Conference on Web Search and Data Mining*, 2020, pp. 519–527.

[76] Y. Yan, L. Liu, Y. Ban, B. Jing, and H. Tong, "Dynamic knowledge graph alignment," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 5, 2021, pp. 4564–4572.

[77] 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, pp. 280–288.

[78] G. Kollias, S. Mohammadi, and A. Grama, "Network similarity decomposition (nsd): A fast and scalable approach to network alignment," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 12, pp. 2232–2243, 2011.

[79] S. Feizi, G. Quon, M. Medard, M. Kellis, and A. Jadbabaie, "Spectral alignment of networks," 2015.

[80] H. Nassar, N. Veldt, S. Mohammadi, A. Grama, and D. F. Gleich, "Low rank spectral network alignment," in *Proceedings of the 2018 World Wide Web Conference*, 2018, pp. 619–628.

[81] Z. Chen, X. Yu, B. Song, J. Gao, X. Hu, and W.-S. Yang, "Community-based network alignment for large attributed network," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 2017, pp. 587–596.

[82] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, "Graph kernels," *The Journal of Machine Learning Research*, vol. 11, pp. 1201–1242, 2010.

[83] 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.

[84] A. Yasar and Ü. V. Çatalyürek, "An iterative global structure-assisted labeled network aligner," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2614–2623.

[85] K. K. Qin, F. D. Salim, Y. Ren, W. Shao, M. Heimann, and D. Koutra, "G-crewe: Graph compression with embedding for network alignment," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 1255–1264.

[86] R. Zafarani and H. Liu, "Connecting users across social media sites: a behavioral-modeling approach," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013, pp. 41–49.

[87] S. Liu, S. Wang, F. Zhu, J. Zhang, and R. Krishnan, "Hydra: Large-scale social identity linkage via heterogeneous behavior modeling," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, 2014, pp. 51–62.

[88] 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.

[89] K. Shu, S. Wang, J. Tang, R. Zafarani, and H. Liu, "User identity linkage across online social networks: A review," *Acm Sigkdd Explorations Newsletter*, vol. 18, no. 2, pp. 5–17, 2017.

[90] M. Yan, J. Sang, T. Mei, and C. Xu, "Friend transfer: Cold-start friend recommendation with cross-platform transfer learning of social knowledge," in *2013 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 2013, pp. 1–6.

[91] A. R. Nelakurthi and J. He, "Finding cut from the same cloth: Cross network link recommendation via joint matrix factorization," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[92] X. Cao, H. Chen, X. Wang, W. Zhang, and Y. Yu, "Neural link prediction over aligned networks," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[93] X. Du, J. Yan, R. Zhang, and H. Zha, "Cross-network skip-gram embedding for joint network alignment and link prediction," *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[94] X. Cao and Y. Yu, "Joint user modeling across aligned heterogeneous sites using neural networks," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2017, pp. 799–815.

[95] Z. Zhu, J. Cao, T. Zhou, and B. Liu, "Understanding user topic preferences across multiple social networks," *arXiv preprint arXiv:2103.07654*, 2021.

[96] O. Kuchaiev, T. Milenković, V. Memišević, W. Hayes, and N. Pržulj, "Topological network alignment uncovers biological function and phylogeny," *Journal of the Royal Society Interface*, vol. 7, no. 50, pp. 1341–1354, 2010.

[97] Y. Sun, W. Wang, N. Wu, W. Yu, and X. Chen, "Anomaly subgraph detection with feature transfer," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 1415–1424.

[98] P. Bindu, P. S. Thilagam, and D. Ahuja, "Discovering suspicious behavior in multilayer social networks," *Computers in Human Behavior*, vol. 73, pp. 568–582, 2017.

[99] H. Zhu, R. Xie, Z. Liu, and M. Sun, "Iterative entity alignment via knowledge embeddings," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

[100] Z. Sun, W. Hu, Q. Zhang, and Y. Qu, "Bootstrapping entity alignment with knowledge graph embedding." in *IJCAI*, vol. 18, 2018, pp. 4396–4402.

[101] Z. Wang, Q. Lv, X. Lan, and Y. Zhang, "Cross-lingual knowledge graph alignment via graph convolutional networks," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018, pp. 349–357.

[102] Y. Cao, Z. Liu, C. Li, J. Li, and T.-S. Chua, "Multi-channel graph neural network for entity alignment," *arXiv preprint arXiv:1908.09898*, 2019.

[103] 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*. ACM, 2014, pp. 1356–1365.

[104] V. Van Vlasselaer, C. Bravo, O. Caelen, T. Eliassi-Rad, L. Akoglu, M. Snoeck, and B. Baesens, "Apate: A novel approach for automated credit card transaction fraud detection using network-based extensions," *Decision Support Systems*, vol. 75, pp. 38–48, 2015.

[105] G. W. Klau, "A new graph-based method for pairwise global network alignment," *BMC bioinformatics*, vol. 10, no. 1, p. S59, 2009.

[106] 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.

[107] H. Tong, C. Faloutsos, and J.-Y. Pan, "Fast random walk with restart and its applications," in *Sixth international conference on data mining (ICDM'06)*. IEEE, 2006, pp. 613–622.

[108] K. B. Petersen, M. S. Pedersen et al., "The matrix cookbook," *Technical University of Denmark*, vol. 7, no. 15, p. 510, 2008.

[109] R. Kondor, N. Teneva, and V. Garg, "Multiresolution matrix factorization," in *International Conference on Machine Learning*. PMLR, 2014, pp. 1620–1628.

[110] 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, 2013.

[111] 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.

[112] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graph evolution: Densification and shrinking diameters," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 1, no. 1, p. 2, 2007.

[113] W. Cohen, P. Ravikumar, and S. Fienberg, "A comparison of string metrics for matching names and records," in *Kdd workshop on data cleaning and object consolidation*, vol. 3, 2003, pp. 73–78.

[114] S. Hashemifar and J. Xu, "Hubalign: an accurate and efficient method for global alignment of protein–protein interaction networks," *Bioinformatics*, vol. 30, no. 17, pp. i438–i444, 2014.

[115] M. Cho, J. Lee, and K. M. Lee, "Reweighted random walks for graph matching," in *European conference on Computer vision*. Springer, 2010, pp. 492–505.

[116] G. Kollias, S. Mohammadi, and A. Grama, "Network similarity decomposition (nsd): A fast and scalable approach to network alignment," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 12, pp. 2232–2243, 2012.

[117] S. Hashemifar, J. Ma, H. Naveed, S. Canzar, and J. Xu, "Modulealign: module-based global alignment of protein–protein interaction networks," *Bioinformatics*, vol. 32, no. 17, pp. i658–i664, 2016.

[118] 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.

[119] T. C. Koopmans and M. Beckmann, "Assignment problems and the location of economic activities," *Econometrica: journal of the Econometric Society*, pp. 53–76, 1957.

[120] A. Myronenko and X. Song, "Point set registration: Coherent point drift," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 12, pp. 2262–2275, 2010.

[121] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE signal processing magazine*, vol. 30, no. 3, pp. 83–98, 2013.

[122] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[123] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 1025–1035.

[124] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[125] V. Sindhwani and D. S. Rosenberg, "An rkhs for multi-view learning and manifold co-regularization," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 976–983.

[126] B. Schölkopf, R. Herbrich, and A. J. Smola, "A generalized representer theorem," in *International conference on computational learning theory*. Springer, 2001, pp. 416–426.

[127] R. I. Kondor and J. Lafferty, "Diffusion kernels on graphs and other discrete structures," in *Proceedings of the 19th international conference on machine learning*, vol. 2002, 2002, pp. 315–322.

[128] A. J. Smola and R. Kondor, "Kernels and regularization on graphs," in *Learning theory and kernel machines*. Springer, 2003, pp. 144–158.

[129] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *International Conference on Machine Learning*. PMLR, 2018, pp. 5453–5462.

[130] J. Zhang and S. Y. Philip, "Integrated anchor and social link predictions across social networks," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[131] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard et al., "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, 2016, pp. 265–283.

[132] 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*, 2016, pp. 765–774.

[133] Y. Yao, H. Tong, G. Yan, F. Xu, X. Zhang, B. K. Szymanski, and J. Lu, "Dual-regularized one-class collaborative filtering," in *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, 2014, pp. 759–768.

[134] Q. Liu, C. Chen, A. Gao, H. H. Tong, and L. Xie, "Varifunnet, an integrated multiscale modeling framework to study the effects of rare non-coding variants in genome-wide association studies: Applied to alzheimer's disease," in *2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, 2017, pp. 2177–2182.

[135] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[136] B. Yang, Y. Lei, J. Liu, and W. Li, "Social collaborative filtering by trust," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 8, pp. 1633–1647, 2016.

[137] 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.

[138] 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.

[139] W. Jin, K. Yang, R. Barzilay, and T. Jaakkola, "Learning multimodal graph-to-graph translation for molecular optimization," *arXiv preprint arXiv:1812.01070*, 2018.

[140] X. Guo, L. Zhao, C. Nowzari, S. Rafatirad, H. Homayoun, and S. M. P. Dinakarrao, "Deep multi-attributed graph translation with node-edge co-evolution," in *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2019, pp. 250–259.

[141] D. Ron, I. Safro, and A. Brandt, "Relaxation-based coarsening and multiscale graph organization," *Multiscale Modeling & Simulation*, vol. 9, no. 1, pp. 407–423, 2011.

[142] R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," *arXiv preprint arXiv:1806.08804*, 2018.

[143] H. Gao and S. Ji, "Graph u-nets," in *international conference on machine learning*. PMLR, 2019, pp. 2083–2092.

[144] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," in *International Conference on Machine Learning*. PMLR, 2019, pp. 3734–3743.

[145] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[146] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[147] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *arXiv preprint arXiv:1903.02428*, 2019.

[148] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "Bpr: Bayesian personalized ranking from implicit feedback," *arXiv preprint arXiv:1205.2618*, 2012.

[149] Z. Yang, W. W. Cohen, and R. Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," *arXiv preprint arXiv:1603.08861*, 2016.

[150] 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, pp. 990–998.

[151] J. Tang, H. Gao, and H. Liu, "mtrust: discerning multi-faceted trust in a connected world," in *Proceedings of the fifth ACM international conference on Web search and data mining*, 2012, pp. 93–102.

[152] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *International conference on machine learning*, 2014, pp. 1188–1196.

[153] 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.

[154] 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.

[155] J. Chen, C. Wang, S. Zhou, Q. Shi, Y. Feng, and C. Chen, "Samwalker: Social recommendation with informative sampling strategy," in *The World Wide Web Conference*, 2019, pp. 228–239.

[156] Z. Yang, M. Ding, C. Zhou, H. Yang, J. Zhou, and J. Tang, "Understanding negative sampling in graph representation learning," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 1666–1676.

[157] M. Armandpour, P. Ding, J. Huang, and X. Hu, "Robust negative sampling for network embedding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 3191–3198.

[158] M. Maruf and A. Karpatne, "Maximizing cohesion and separation in graph representation learning: A distance-aware negative sampling approach," in *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*. SIAM, 2021, pp. 271–279.

[159] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *Thirty-Second AAAI conference on artificial intelligence*, 2018.

[160] 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*, 2014, pp. 701–710.

[161] 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*, 2016, pp. 855–864.

[162] S. Rendle and C. Freudenthaler, "Improving pairwise learning for item recommendation from implicit feedback," in *Proceedings of the 7th ACM international conference on Web search and data mining*, 2014, pp. 273–282.

[163] Y. Liu, F. Shang, H. Cheng, J. Cheng, and H. Tong, "Factor matrix trace norm minimization for low-rank tensor completion," in *Proceedings of the 2014 SIAM International Conference on Data Mining*. SIAM, 2014, pp. 866–874.

[164] B. Recht, "A simpler approach to matrix completion." *Journal of Machine Learning Research*, vol. 12, no. 12, 2011.

[165] J. D. Rennie and N. Srebro, "Fast maximum margin matrix factorization for collaborative prediction," in *Proceedings of the 22nd international conference on Machine learning*, 2005, pp. 713–719.

[166] C. Ding, X. He, and H. D. Simon, "On the equivalence of nonnegative matrix factorization and spectral clustering," in *Proceedings of the 2005 SIAM International Conference on Data Mining*. SIAM, 2005, pp. 606–610.

[167] J. J. McAuley and J. Leskovec, "Learning to discover social circles in ego networks." in *NIPS*, vol. 2012. Citeseer, 2012, pp. 548–56.

[168] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," *Knowledge and Information Systems*, vol. 42, no. 1, pp. 181–213, 2015.

[169] U. Brandes, "On variants of shortest-path betweenness centrality and their generic computation," *Social Networks*, vol. 30, no. 2, pp. 136–145, 2008.

[170] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU Press, 2012, vol. 3.

[171] Z. Xu, S. Zhang, Y. Xia, L. Xiong, and H. Tong, "Ranking on network of heterogeneous information networks," in *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 2020, pp. 848–857.

[172] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *International conference on machine learning*. PMLR, 2019, pp. 6861–6871.

[173] M. R. Khan and J. E. Blumenstock, "Multi-gcn: Graph convolutional networks for multi-view networks, with applications to global poverty," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 606–613.

[174] C. Park, J. Han, and H. Yu, "Deep multiplex graph infomax: Attentive multiplex network embedding using global information," *Knowledge-Based Systems*, vol. 197, p. 105861, 2020.

[175] Y. Bai, H. Ding, S. Bian, T. Chen, Y. Sun, and W. Wang, "Simgnn: A neural network approach to fast graph similarity computation," in *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, 2019, pp. 384–392.

[176] Y. Zuo, G. Liu, H. Lin, J. Guo, X. Hu, and J. Wu, "Embedding temporal network via neighborhood formation," in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, 2018, pp. 2857–2866.

[177] Y. Lu, X. Wang, C. Shi, P. S. Yu, and Y. Ye, "Temporal network embedding with micro-and macro-dynamics," in *Proceedings of the 28th ACM international conference on information and knowledge management*, 2019, pp. 469–478.

[178] X. Chang, X. Liu, J. Wen, S. Li, Y. Fang, L. Song, and Y. Qi, "Continuous-time dynamic graph learning via neural interaction processes," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 145–154.

[179] F. Manessi, A. Rozza, and M. Manzo, "Dynamic graph convolutional networks," *Pattern Recognition*, vol. 97, p. 107000, 2020.

[180] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein, "Temporal graph networks for deep learning on dynamic graphs," *arXiv preprint arXiv:2006.10637*, 2020.

[181] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," *arXiv preprint arXiv:1810.05997*, 2018.

[182] P. Panzarasa, T. Opsahl, and K. M. Carley, "Patterns and dynamics of users' behavior and interaction: Network analysis of an online community," *Journal of the American Society for Information Science and Technology*, vol. 60, no. 5, pp. 911–932, 2009.

[183] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: densification laws, shrinking diameters and possible explanations," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, 2005, pp. 177–187.

[184] N. Malod-Dognin and N. Pržulj, "L-graal: Lagrangian graphlet-based network aligner," *Bioinformatics*, vol. 31, no. 13, pp. 2182–2189, 2015.

[185] J. Kang, J. He, R. Maciejewski, and H. Tong, "Inform: Individual fairness on graph mining," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 379–389.

[186] P. Li, Y. Wang, H. Zhao, P. Hong, and H. Liu, "On dyadic fairness: Exploring and mitigating bias in graph connections," in *International Conference on Learning Representations*, 2020.

[187] Y. Dong, J. Kang, H. Tong, and J. Li, "Individual fairness for graph neural networks: A ranking based approach," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 300–310.