Title
Practical and Scalable Methods for Information Network Analysis and Mining

Author
Yu, Wenchao

Publication Date
2019

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA
Los Angeles

Practical and Scalable Methods for Information Network Analysis and Mining

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Wenchao Yu

2019

ABSTRACT OF THE DISSERTATION

Practical and Scalable Methods for
Information Network Analysis and Mining

by

Wenchao Yu
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 2019
Professor Wei Wang, Chair

The problem of information network analysis has gained increasing attention in recent years, because most objects and data in the real world are interconnected, forming complex networks. One challenging aspect of network analysis is that they are inherently resistant to parametric modeling, which allows us to truly express the vertices and edges in the network as vectors or functions of time. This is because, unlike multi-dimensional data, the edges in the network reflect interactions among vertices, and it is difficult to independently model them without taking into account their correlations and interactions with neighboring vertices or edges. This thesis presents a combination of the methods and applications in *static network* analysis and *evolutionary network* analysis, which is trying to analyze and model the structure and evolution in networks.

On the analysis of static network side, we develop methods to learn the network representations with adversarially regularized autoencoders, which learns smoothly regularized vertex representations that well capture the network structure through jointly considering both locality-preserving and global reconstruction constraints. And applications in community detection are designed to detect communities which have the maximum competition intensity score in an advertiser-keyword bipartite network.

On the analysis of evolutionary network side, we show that it is indeed possible to represent the network structure purely as a function of time with the use of temporal matrix factorization, in which the entries are parameterized by time. This opens the possibility of using the approach

for a wide variety of evolutionary network analysis problems, such as temporal link prediction. It can also be utilized to model co-evolution across multiple networks by decomposing the adjacency matrix of each co-evolving network into a product of network-independent factors and a set of network-specific time-dependent factors. Applications in temporal link prediction and online anomaly detection are proposed, in which the low-dimensional representations of networks can be learned and updated to capture evolutionary network structures.

The dissertation of Wenchao Yu is approved.

Junghoo Cho

Carlo Zaniolo

Yingnian Wu

Wei Wang, Committee Chair

University of California, Los Angeles

2019

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

VITA

2011	B.Eng. in Computer Science, Wuhan University, Wuhan, China
2014	M.S. in Computer Science, University of Chinese Academy of Sciences, Beijing, China
2015, 2016	Research Intern (Summer), IBM Thomas J. Watson Research Center, Yorktown Heights, NY
2017, 2018	Research Intern (Summer), NEC Labs America, Princeton, NJ
2014–2019	Research Assistant, Computer Science Department, UCLA, Los Angeles, CA

PUBLICATIONS

1. Wenchao Yu, Wei Cheng, Charu C Aggarwal, Kai Zhang, Haifeng Chen and Wei Wang. Netwalk: A Flexible Deep Embedding Approach for Anomaly Detection in Dynamic Networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2018.

2. Wenchao Yu, Cheng Zheng, Wei Cheng, Charu C Aggarwal, Dongjin Song, Bo Zong, Haifeng Chen and Wei Wang. Learning Deep Network Representations with Adversarially Regularized Autoencoders. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2018.

3. Wenchao Yu, Charu C Aggarwal, and Wei Wang. Modeling co-evolution across multiple networks. In *Proceedings of the SIAM International Conference on Data Mining (SDM)*. SIAM, 2018.

4. Wenchao Yu, Wei Cheng, Charu C Aggarwal, Haifeng Chen, and Wei Wang. Link prediction with spatial and temporal consistency in dynamic networks. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3343–3349, 2017.

5. Wenchao Yu, Charu C Aggarwal, and Wei Wang. Temporally factorized network modeling for evolutionary network analysis. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining (WSDM)*, pages 455–464. ACM, 2017.

6. Wenchao Yu, Ariyam Das, Justin Wood, Wei Wang, Carlo Zaniolo, and Ping Luo. Max-intensity: Detecting competitive advertiser communities in sponsored search market. In *International Conference on Data Mining (ICDM)*, pages 569–578. IEEE, 2015.

7. Wenchao Yu, Fuzhen Zhuang, Qing He, and Zhongzhi Shi. Learning deep representations via extreme learning machines. *Neurocomputing*, 149:308–315, 2015.

8. Wenchao Yu, Guangxiang Zeng, Ping Luo, Fuzhen Zhuang, Qing He, and Zhongzhi Shi. Embedding with autoencoder regularization. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD)*, pages 208–223. Springer, 2013.

CHAPTER 1

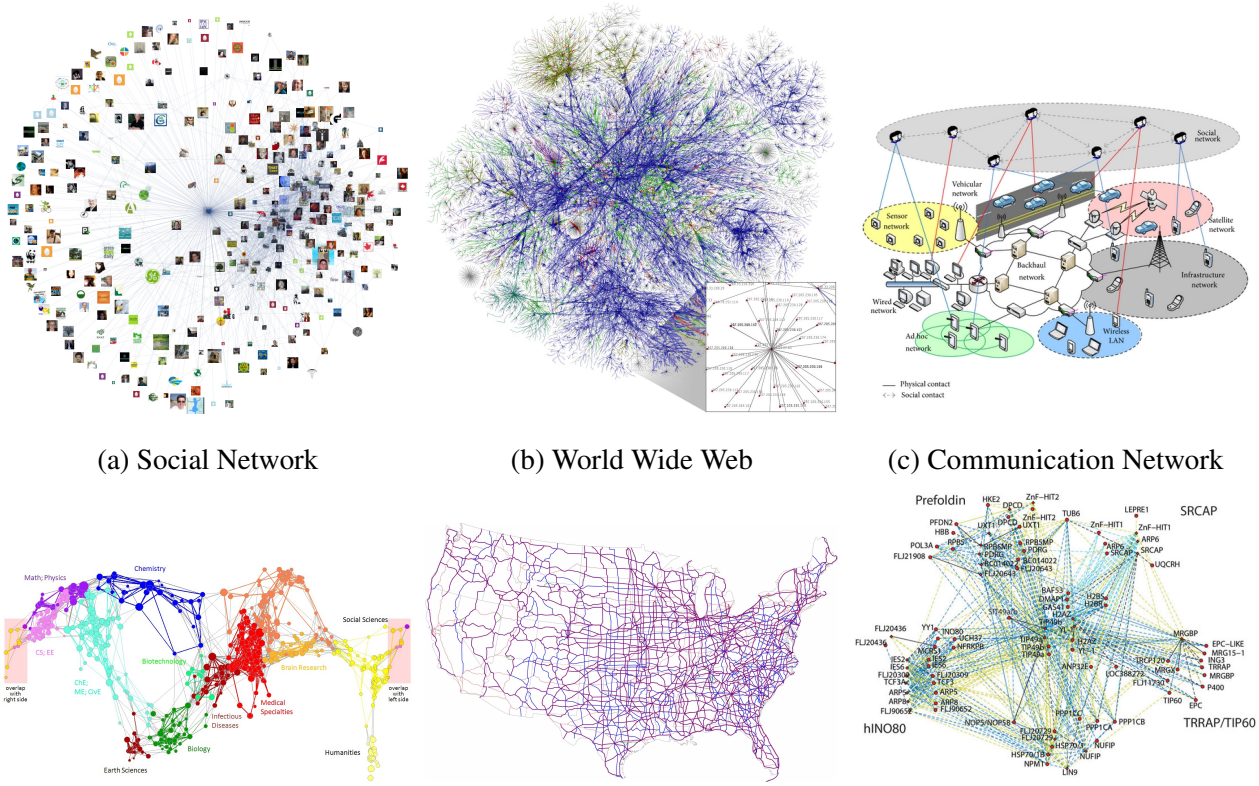
Introduction

Information networks have emerged as a powerful conceptual paradigm in science and engineering, which appear in a wide variety of researches, ranging across social networks, communication networks, citation networks, etc., as shown in Figure 1.1. The reason for this unprecedented attention to information networks is twofold. On the one hand, this attention is driven by *applications* in areas such as biological and social science, which are trying to gain a deeper understanding of the roles that the inter-elemental interactions play in the networked data. On the other hand, advanced technological *methods* have facilitated the ability to analyze the complex networks. This thesis will introduce various methods and applications with different forms of information networks. In particular, we will categorize these networks into two groups: *static networks* and *evolutionary networks*, as described below,

Static Networks: In these networks, the edges and vertices are static, that is, the edge and vertex sets will not be varying over time. This is, for example, the situation in road network analysis. We can assume that network is fixed after being established.

Evolutionary Networks: These networks are changing as a function of time. Here edge and vertex sets are time varying, either by adding or removing edges or vertices over time. For examples, in social networks, people make and lose friends over time, thereby creating and destroying edges, and some people become part of new social networks or leave their networks, changing the vertices in the network.

The analysis of static and evolutionary information networks has been attracting many research interests with its enormous potential in mining useful information which benefits the downstream tasks such as link prediction, community detection and anomaly detection. A basic premise in information network analysis is that the structure and attributes of the network influence the prop-



(a) Social Network (b) World Wide Web (c) Communication Network

(d) Citation Network (e) Road Network (f) Protein Interaction Network

Figure 1.1: Network examples

Social Network: https://williamjturkel.net/2011/08/02/social-network-analysis-and-visualization/

World Wide Web: http://rtfm.readthedocs.io/en/latest/security/network.html

Communication Network: https://www.hindawi.com/archive/2013/972352/

Citation Network: http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0039464

Road Network: https://en.wikipedia.org/wiki/United_States_Numbered_Highway_System

Protein Interaction Networks: http://research.stowers.org/proteomics/ProtNetAnal.html

erties exhibited at the network level. Next, we will describe the problems in information network analysis that motivated our research.

1.1 Motivation

Ultimately we search for methods to learn interesting representations that let us characterize the underlying network structures with both static and dynamic settings. Then we design applications that take advantages of the identified structural network representations by the learning methods. We describe the motivations below regarding to different forms of information networks.

1.1.1 PART I – Static Network Analysis

To analyze the static network data, one fundamental problem is to learn a low-dimensional vector representation for each vertex, such that the network structure is preserved in the learned vector space. For this problem, there are two major challenges:

- *preservation of complex structure property*. The objective of network embedding is to train a model to "fit" the training networks, that is, to preserve the structure property of networks. However, the latent structure of the network is too complex to be portrayed by an explicit form of probability density which can capture both the local network neighborhood information and global network structure.

- *sparsity of network sampling*. Current research on network embedding employs network sampling techniques, including random walk sampling, breadth-first search etc., to derive vertex sequences as training datasets. However, the sampling strategy suffers from the data sparsity problem since the total amount of vertex sequences is usually very large in real networks and it is often intractable to enumerate all. Subsequently, learning on a sparse sample set tends to produce an overly complex model to explain the sampled dataset, which eventually causes overfitting.

Therefore, carefully designed regularizations are still desirable for improving generalization of the learned network representations. Moreover, most network embedding models with deep architectures usually do not consider the order of the vertices in the sampled vertex sequences. Thus, the information of proximity orders cannot be well considered.

On the application side, this thesis focuses on community detection task. In a sponsored search market, different advertisers, bidding on the same set of keywords, compete against each other for their ad slot in the search results. A common question to answer is *"what is the best way to measure the competition among different advertisers in the bi-partite advertiser-keyword network?"*. This is a very important question from the search providers' perspective, as understanding the intensity of competition among different advertisers can help them monitor a sector better. However, not much work has been conducted in identifying competitive advertiser communities.

1.1.2 PART II – Evolutionary Network Analysis

In recent years, a significant amount of work has been done on the area of evolutionary network analysis, which examines various problems in the context of network evolution. Some examples of such problems are as follows:

- Based on the trends in the past, which links are most likely to be received at a future point in time? How does the likelihood change with increasing value of time. Note that this is a more refined problem than traditional link prediction, in which one simply predicts the links based on a static state of the network.

- How do communities evolve over time? Which communities grow, and which ones shrink? Which ones are expected to grow in the future? Numerous works have been proposed in this context, although none of these methods fully capture the evolving nature of the underlying network.

- One would like to predict surprising or anomalous events in different regions of the networks. These could represent sudden regions of change, or other structural changes in the network.

Although many individual solutions exist for these problems, a broader question is whether we can directly characterize the structure of the network as a function of time. The ability to characterize the structure of the network as a function of time is crucial in using it in different application settings such as network co-evolution, because such a characterization can capture very rich information about the structure of the underlying network.

On the application side, this thesis focuses on online anomaly detection and temporal link prediction tasks. Recently, network embedding has attracted significant interest and shown promising results, in particular towards obtaining desired low-dimensional network representations that better preserve the neighborhood information. The structure preserving property of the network embedding makes it particularly suitable for anomaly detection tasks, by examining the similarity between vertices/edges in the latent representation. For example, vertices staying far away from the majority clusters in the multidimensional latent space will very likely indicate certain types of anomalies, which can be detected conveniently through dynamic clustering algorithms. However,

existing methods for network embedding can not update the representation dynamically as new vertices or edges keep feeding, and thus may not be perfectly suitable for anomaly detection in a dynamic environment. In case of a rapidly evolving network, the problem can be even more challenging. It is therefore highly desirable to design an effective and especially efficient embedding algorithm that is capable of fast, real-time detection with bounded memory usage. Regarding to the temporal link prediction task, the key factors to link prediction task in temporal networks are spatial and temporal consistencies, which mean: 1) a node has higher probability to form a link with a nearby node than with a remote node in the near future; 2) a network usually evolves smoothly over time. Existing approaches, however, seldom unify these two factors to strive for the spatial and temporal consistency in the dynamic network.

1.2 Thesis Outline

This thesis addresses a number of important questions regarding the methods and applications of information network analysis under static and evolutionary settings, which have been categorized into two parts: *static network analysis* and *evolutionary network analysis*. Table 1.1 gives the overall structure of our research with the mapping to the chapters of this thesis. We break each part into *methods* and *applications* sections in the thesis.

1.2.1 PART I – Static Network Analysis

In this part we focus on the methods and applications on *static* network analysis. To analyze network data, one fundamental problem is to learn a low-dimensional vector representation for each vertex, such that the network structure is preserved in the learned vector space. In Chapter 3, we propose NETRA, a novel model to learn the network representations with adversarially regularized autoencoders. NETRA exhibits desirable properties that a network embedding model requires: 1) *structure property preservation*, NETRA leverages recurrent neural network as an encoder to capture the neighborhood information among vertices in each sequence sampled from the network. Additionally, the model is also trained simultaneously with the locality-preserving constraint. 2) *generalization capability*, the generalization capability requires a network embedding model to

Table 1.1: Structure of the thesis with references to the chapters

Thesis Parts	Methods	Applications
PART I: Static Network Analysis	Chapter 3	Chapter 4
	NETRA (KDD'18)	MAXINTENSITY (ICDM'15)
PART II: Evolutionary Network Analysis	Chapter 5	Chapter 6
	TMF (WSDM'17)	NETWALK (KDD'18)
	COEVOL (SDM'18)	LIST (IJCAI'17)

generalize well on unseen vertex sequences which follow the same distribution as the population. The generative adversarial training process enables the proposed model to learn smoothly regularized representations without pre-defining an explicit density distribution which overcomes the sparsity issue from the input sequences of vertices.

On the application side, Chapter 4 introduces a novel approach to detect competitive communities in a weighted bi-partite network formed by advertisers and their bidding keywords. The proposed approach is based on an advertiser vertex metric called intensity score, which takes the following two factors into consideration: the competitors that bid on the same keywords, and the advertisers' consumption proportion within the community. Thereafter, we propose a community detection algorithm MAXINTENSITY which detects highly competitive communities by maximizing the total intensity score within each community.

1.2.2 PART II – Evolutionary Network Analysis

Evolving networks are common in a wide variety of settings because of the importance of different types of dynamic networks and social streams. In this part we present the methods and applications on *evolutionary* network analysis. In Chapter 5 we show that it is indeed possible to express the edges/vertices in the network as functions of time with the use of temporal matrix factorization (TMF), in which the entries are parameterized by time. This opens the possibility of using the approach for a wide variety of temporal network analysis problems, such as predicting future trends in structures, predicting links, and node-centric anomaly detection. This flexibility is because of the

general way in which the approach allows us to express the structure of the network as a function of time. In practice, a significant amount of information is encoded in the evolution of multiple networks with respect to one another. With this observation, we propose CoEvol to model co-evolution across multiple networks with *shared temporal matrix factorization*, which decomposes the adjacency matrix of each co-evolving network into a product of network-independent shared factor and a set of network-specific temporal factors, and impose a non-negativity constraint on the factors for greater interpretability. Even though we provide more general models, our results show that their specific instantiations to different prediction tasks perform better than state-of-the-art techniques, thereby demonstrates the generality and effectiveness of the TMF and CoEvol methods.

On the application side, Chapter 6 introduces two applications, namely anomaly detection and link prediction, under the dynamic setting. Given an evolutionary network, it is crucial to detect structural anomalies, such as vertices and edges whose "behaviors" deviate from underlying majority of the network, in a real-time fashion. We propose a novel approach, NetWalk, for anomaly detection in dynamic networks by learning network representations which can be updated dynamically as the network evolves. The vertices of the dynamic network are encoded to vector representations by *clique embedding*, which jointly minimizes the pairwise distance of vertex representations of each walk derived from the dynamic networks, and the deep autoencoder reconstruction error serving as a global regularization. The vector representations can be computed with constant space requirements using *reservoir sampling*. On the basis of the learned low-dimensional vertex representations, a clustering-based technique is employed to incrementally and dynamically detect network anomalies. Link prediction in dynamic networks has also attracted tremendous research interests. There are two key factors: 1) a node is more likely to form a link in the near future with another node within its close proximity, rather than with a random node; 2) a dynamic network usually evolves smoothly. We develop a temporal link prediction model, LIST, to strive for the spatial and temporal consistency in evolutionary networks. LIST characterizes the network dynamics as a function of time, which integrates the spatial topology of network at each timestamp and the temporal network evolution.

1.3 Contributions

The thesis focuses on static (PART I) and time evolving networks (PART II), where each of them is composed of two chapters: methods and applications, as summarized in Table 1.1. In PART I, we proposed NETRA to learn network representations, and MAXINTENSITY to detect competitive communities under static setting. And in PART II, we propose TMF and COEVOL for evolutionary network analysis based on time-dependent matrix factorization. Meanwhile, NETWALK and LIST are developed for online anomaly detection and temporal link prediction.

Static Network Analysis:

- We propose a novel deep network embedding model with adversarially regularized autoencoders, NETRA, to learn vertex representations by jointly minimizing locality-preserving loss and global reconstruction error using generative adversarial training process. The resultant representations are robust to the sparse inputs derived from the network. Extensive experiments are conducted on tasks of network reconstruction, link prediction and vertex classification using real-world information networks. Experimental results demonstrate the effectiveness and efficiency of NETRA.

- We propose a new metric *intensity score* to represent the competition in an advertiser-keyword network (modeled as a weighted bi-partite graph), and we propose *Max-Intensity* algorithm which is based on *intensity score*, to detect competitive communities within a sector. We extend the concepts of market concentration measures from retail markets to calculate the competition in a sponsored search marketplace.

Evolutionary Network Analysis:

- We developed TMF, a novel temporal matrix factorization model for dynamic network analysis. TMF has the advantage of significant generality in addressing various temporal applications because of its ability to explicitly represent the network as a function of time.

- We proposed COEVOL to utilize a *shared temporal matrix factorization* framework to model co-evolution across multiple networks, which decomposes the adjacency matrix of each co-

evolving network into a product of network-independent shared factor and a set of network-specific temporal factors, and impose a non-negativity constraint on the factors for greater interpretability.

- We proposed NETWALK to detect anomalies in dynamic networks, by learning faithful network representations which can be updated dynamically as the network evolves over time based on *deep* neural network embedding and *reservoir sampling*. Quantitative validation on anomaly detection task showed that NETWALK is computationally efficient and outperforms state-of-the-art techniques.

- We proposed a novel temporal link prediction model, LIST, for dynamic networks which simultaneously consider the network structure at each timestamp and the evolutionary pattern across all timestamps. We leverage network propagation constraint which ensures that the connected vertices will have similar feature vectors. This "locality-preserving" property captures the spatial structure of networks at each timestamp. The feature vector of each vertex can be learned via time-dependent matrix factorization across all recent timestamps, which reveals the temporal structure of networks.. Extensive experiments show that the LIST model outperforms the state-of-the-art techniques.

CHAPTER 2

Overview and Survey

In this chapter, we aim to provide readers with an overview of information network analysis methods and applications. We begin with a review of the basic concepts and terminology used in this thesis. We then survey the literatures on static and evolutionary network analysis, as well as the applications including link prediction, community detection and anomaly detection.

2.1 Static Network Analysis

Matrices are the most straightforward representation of a network. Earlier work has focused on factorization of different matrix representations, including but not limited to adjacency matrix [Roweis and Saul, 2000], Laplacian matrix [Tang and Liu, 2011], and transition probability matrix [Cao et al., 2015]. Matrix factorization approaches usually have time complexity of $\mathcal{O}(|V|^2)$, which restricts their applications. Thus one challenge facing matrix factorization is its poor scalability as the data volume explodes with time, as demonstrated in [Zhou et al., 2017] on a real-world network of Alibaba Group, which has 290 million vertices and 18 billion edges.

Recently, we have witnessed the emergence of random walk based methods [Dong et al., 2017, Perozzi et al., 2014b, Grover and Leskovec, 2016], inspired by the success of natural language processing [Perozzi et al., 2014b]. These models build connections between network structure and natural language. The training input of these algorithms changes from matrices to sentence-like vertex sequences generated by random walks among connected vertices. The skip-gram algorithm [Mikolov et al., 2013b] maximizes the co-occurrence probability among the vertices within a certain window in a random walk. DeepWalk [Perozzi et al., 2014b] obtains effective embeddings using truncated random walks. node2vec [Grover and Leskovec, 2016] extends the model

with flexibility between homophily and structural equivalence [Zhang et al., 2016]. These last two methods motivate the study of network embedding taking advantage of language models. Additionally, a wide variety of models have been proposed for attributed networks [Huang et al., 2017a, Yang et al., 2015], which show that jointly learning network representations with network topology information and vertex attributes enhance the performance on various tasks. The text-associated DeepWalk [Yang et al., 2015] incorporates text features of vertices into network representation learning under the framework of matrix factorization. Similarly, an accelerated attributed network embedding learns an effective unified embedding representation by incorporating node attribute proximity into network embedding [Huang et al., 2017a].

However, these matrix factorization based shallow models are not enough for the complicated networks. Deep learning embedding models [Tian et al., 2014, Cao et al., 2016, Wang et al., 2016, Gao and Huang, 2018, Bojchevski and Günnemann, 2018] have been applied to solve the network embedding problem. Autoencoder based approaches [Wang et al., 2016, Cao et al., 2016] were proposed, utilizing its ability of learning highly non-linear properties. By carefully constructing the learning objective, [Wang et al., 2016] preserves the first and second proximity of networks which delivers the state-of-the-art performance. Recent works on graph convolutional networks [Defferrard et al., 2016, Kipf and Welling, 2016] have demonstrated effective convolution operation on network data. Inductive and unsupervised GraphSAGE [Hamilton et al., 2017] leverages vertex features [Huang et al., 2017b] and aggregates features among vertex neighborhood.

The rapid advances in deep learning research in last decades have provided novel methods for studying highly non-linear data. One of such models is the generative adversarial networks (GANs) [Goodfellow et al., 2014] which has achieved great success in generating and learning the latent presentation of high dimensional data, such images [Radford et al., 2015]. There have been several successful attempts [Gulrajani et al., 2017, Rajeswar et al., 2017, Kim et al., 2017] of implementing GANs on discrete structures, such as text and discrete images, which inspired us to investigate network representation learning using GANs. Using GANs to learn the representation of discrete contents like natural languages and social networks remains a challenging problem due to the difficulty in back-propagation through discrete random variables. Recent work on GANs such as GraphGAN [Wang et al., 2018] and ANE [Dai et al., 2017] for discrete data is

either though the use of discrete structures [Yu et al., 2017a, Che et al., 2017] or the improved autoencoders [Kim et al., 2017].

2.2 Evolutionary Network Analysis

Evolutionary networks [Aggarwal and Subbian, 2014, Ranshous et al., 2015] have recently found increasing importance because of their numerous applications for trend detection in the networks. Numerous methods have been proposed in the context of temporal link prediction [Sarkar et al., 2012, Murata and Moriyasu, 2007, Zhao et al., 2015], dynamic community detection [Tang et al., 2008, Chi et al., 2007, Backstrom et al., 2006, Gupta et al., 2011], compression [Sun et al., 2007], mixed membership modeling [Fu et al., 2009, Rossi et al., 2013], and anomaly detection [Yu et al., 2013a, Ide and Kashima, 2004]. The approach in [Rossi et al., 2013] uses non-negative matrix factorization to *extract features* in a sequence of graphs which is different from the structural factorization model in the paper. The applications in [Rossi et al., 2013] are implicitly regulated by the nature of the node features that are extracted, and cannot fully characterize the structure of the network over time or directly express the network structure as a function of time, once the features have been extracted. The fully parameterized model in this paper is more general, and it can be used to reconstruct the approximate future structure in the network at any point in time. Recently, some interest has been focused on the use of r^{th} order tensors [Dunlavy et al., 2011, Sun et al., 2006] for expressing dynamic networks. Tensors are, however, inherently designed for the case when the other $(r-2)$ dimensions of interest (than source node and destination node) are discrete variables rather than continuous; time is a continuous variable. Although some temporal applications have been designed with tensors [Dunlavy et al., 2011], by treating time as a discrete variable, the applicability of these methods to express the network as a continuous function of time is limited. Temporal matrix factorization has recently been used successfully in the context of collaborative filtering [Koren, 2010]. In this thesis, we explore temporal matrix factorization in the context of network-centric applications. We show that a significant number of evolutionary network applications can be addressed with the use of the factorization framework.

The increasing number of networks that encountered in temporal settings has also gained atten-

tion to the study of co-evolutionary dynamics between networks [Zheleva et al., 2009, Sun et al., 2014, Farajtabar et al., 2015, Ellwardt et al., 2012]. Among all these proposed models, they focus on specific types of networks such as social and affiliation networks [Zheleva et al., 2009], star networks (a special type of heterogeneous networks) [Sun et al., 2014], or jointly learn information diffusion and network evolution [Farajtabar et al., 2015]. However, these models are designed based on the features or topology of certain networks, which limit their potential to capture the correlation information among multiple co-evolution networks.

2.3 Applications

The past decades have given rise to significant advances in the development and application of methods for analyzing information networks. The main applications including link prediction, community detection and anomaly detection. This section presents a review of these applications in network analysis.

2.3.1 Link Prediction

Link prediction problem can be generally divided into two distinct categories: *structural* link prediction and *temporal* link prediction [Menon and Elkan, 2011]. The structural link prediction problem, which only considers a single network structure as the input, predicts the possible unobserved links within the same network [Liben-Nowell and Kleinberg, 2007, Lichtenwalter et al., 2010]. Temporal link prediction models, on the other hand, analyze the evolution pattern of a sequence of networks over time [Sarkar et al., 2012, Dunlavy et al., 2011, Li et al., 2014, Zhu et al., 2016, Rahman and Al Hasan, 2016]. The probabilistic nonparametric link prediction model can be used to predict the linkage possibility of two nodes only based on the similarity of their local neighborhoods [Sarkar et al., 2012]. One can also extend a local probabilistic model [Wang et al., 2007] based on maximum entropy with temporal information of the past interactions [Tylenda et al., 2009]. Experimental results show this time-awareness model outperforms the simple time-invariant baseline predictors. Other than the aforementioned statistical approaches, we can also leverage tensor decomposition technique to address the temporal link prediction problem effectively [Ermiş et al., 2015,

Dunlavy et al., 2011]. The global network structure has also been considered in the temporal link prediction task [Gao et al., 2011]. It aggregates weighted link matrices, and learns the model with graph regularization. Most of the existing link prediction models only predict the existence of links, i.e., $a(i, j) \in \{0, 1\}$, where 0 represents the absence of a link, and 1 represents the existence of a link. The weights of links are rarely taken into account [Lü and Zhou, 2009].Recently, some interest has been focused on the use of temporal matrix factorization technique for expressing dynamic networks [Yu et al., 2017b]. However, it only explores the network evolving pattern across the dynamic network, but ignores the network propagation within each single timestamp.

2.3.2 Community Detection

Traditional methods involve clustering or partitioning the graph in a way as to discover communities. Often they require parameters which must be specified a priori, such as k-means clustering [MacQueen et al., 1967], or are highly dependent on a defined similarity measure as in hierarchical clustering [Hastie et al., 2009]. Hierarchical clustering, which also has the advantage of detecting the hierarchical structure of communities, can be split into two categories: agglomerative, and divisive algorithms. Agglomerative algorithms are a bottom up approach where clusters are merged together using a similarity metric in a series of iterations. The approach of divisive algorithms is top down starting with large clusters and then iteratively breaking them apart. A well known and widely used community detection metric is modularity [Newman, 2006b], which is a comparison of edges inside a cluster to the expected number of randomly distributed edges. Methods using this quality function attempt to maximize modularity of partitions in a graph. Since obtaining the maximal modularity has been shown to be a NP-Complete problem [Brandes et al., 2006], approximation techniques must be used. This set of techniques include greedy, simulated annealing, extremal optimization, and spectral optimization techniques. Although the techniques can be quite good at estimating the maximal modularity, the metric may suffer some shortcomings in detecting "good" partitions [Fortunato and Barthélemy, 2007]. Other methods such as spin models, random walks, and synchronizing can be described as a set of stochastic algorithms, and can be used for detecting communities. Spin models originated in statistical mechanics [Wu, 1982], and consist of a system of spins in different states. By apply-

ing these spin variables to vertices of a graph then clusters of the graph can be discovered by identifying like-valued spin clusters. In random walks [Hughes, 1996] it is likely that a random walker will spend more time inside a community than crossing between different communities. One approach which exploits this assumption to find communities is to define a distance measure based on random walks [Zhou and Lipowsky, 2004] and then finding "close" vertices. The application to community detection involves placing oscillators at vertices, which are initially in random phases, and detecting which oscillators synchronize first [Arenas et al., 2006]. To address the resolution limit problem of modularity-based methods [Newman, 2006b], recently Duan et al. proposed an approach to incorporate correlation analysis into the modularity-based method by subtly reformatting their math formulas and objective functions [Duan et al., 2014]. Recent studies also include *Permanence*, a vertex-based metric proposed to identify the community structures in the graph [Chakraborty et al., 2014], aiming at estimating the internal and external connectivity of the vertex to individual communities. Another approach based on heat kernel is to compute this graph diffusion and use that to study the communities that it produces [Kloster and Gleich, 2014]. Besides, to identify the members of a potential but unlabeled community in large-scale social network, one can apply seed expansion to find remaining community members outside current dataset given sample community members [Kloumann and Kleinberg, 2014].

2.3.3 Anomaly Detection

Anomaly detection (a.k.a. outlier detection) has been extensively studied in the context of multi-dimensional data [Cheng et al., 2016, Aggarwal, 2013, Gupta et al., 2014a] and structured information networks [Gao et al., 2010, Akoglu et al., 2015, Ranshous et al., 2015, Akoglu et al., 2010, Eberle and Holder, 2007, Gupta et al., 2014b], including networks with a rich set of attributes and other side information [Perozzi and Akoglu, 2016, Perozzi et al., 2014a, Sun et al., 2005]. Massive networks arise in many applications such as social media and public health, thus numerous algorithms have been developed for processing networks in the stream model [Aggarwal et al., 2010, Aggarwal et al., 2011, Ranshous et al., 2016]. In this section, we briefly review anomaly detection algorithms in dynamic networks, as well as network embedding techniques.

The anomalies of static networks can be generally divided into three distinct types: anomalous vertices, anomalous edges and anomalous subgraphs or communities [Ranshous et al., 2015]. A number of anomaly detection methods on networks are designed for vertex or edge based detection tasks [Aggarwal, 2013, Akoglu et al., 2010, Cheng et al., 2016, Eberle and Holder, 2007]. One of the early methods [Eberle and Holder, 2007] employed the minimum description length principle to establish the normative pattern, then used a trail of pattern expansion to discover particular anomalous instances. Subsequent work [Akoglu et al., 2010] discovered several rules in density, weights, ranks and eigenvalues that derived from the neighborhood sub-networks, which is referred to as the egonet, and proposed a scalable and un-supervised method for anomalous vertex detection. Recent work [Cheng et al., 2016] adopted network diffusion to detect causal anomalies in invariant network. This method also focused on static networks, though its variants can achieve a temporal smoothing effect. Anomalous subgraphs were defined as the subgraphs containing less common substructures [Noble and Cook, 2003]. This work used the Subdue system [Holder et al., 1994] to discover patterns within graphs, and then anomalies occur if the pattern is not infrequent. There are also some work proposed to detect community outlier [Gao et al., 2010, Gupta et al., 2014b], defined as the instance with non-conforming patterns compared with other members in the same community. Work also exists in studying how to detect anomalies for attributed graphs [Perozzi and Akoglu, 2016, Perozzi et al., 2014a, Sun et al., 2005]. The attributes include user preference [Perozzi et al., 2014a], and neighborhoods in graphs [Sun et al., 2005, Perozzi and Akoglu, 2016]. These methods are proposed for static graphs, and not generally applicable to network streams.

In streaming networks, a number of methods perform anomaly detection in the context of edge streams [Ranshous et al., 2015, Aggarwal et al., 2011, Gupta et al., 2014a, Ranshous et al., 2016]. For instance, GOutlier [Aggarwal et al., 2011] introduced a structural connectivity model to define anomalies, and proposed a reservoir sampling method to maintain structural summaries of the underlying graph streams. The anomalies can then be identified as those graph objects which contain unusual bridging edges. The recent work [Ranshous et al., 2016] proposed an anomaly detection method based on edge scoring. The score of an incoming edge was based on historical evidence and vertex neighborhood. There is a new type of anomalies in graph streams: anomalous graph

snapshots [Gupta et al., 2014a, Manzoor et al., 2016, Ranshous et al., 2015]. The StreamSpot introduced a new similarity function to compare two heterogeneous graphs based on their relative frequency of local substructures, and leveraged a centroid-based clustering methods to capture the normal behaviors [Manzoor et al., 2016]. Graph stream clustering algorithms like GMicro [Aggarwal et al., 2010] created sketch-based micro clusters which using a hash-based compression of the edges to a lower-dimensional domain space in order to reduce the size of representation. Variants include graph streams with attributes [McConville et al., 2015, Zhao and Yu, 2013]. The communities evolve across snapshots. The evolutionary community anomalies can be defined as those objects which evolve in a very different way rather than following the community change trends [Gupta et al., 2012a, Gupta et al., 2012b].

Part I

Static Network Analysis

CHAPTER 3

Static Network Analysis: Methods

The analysis of complex networks has recently received considerable attentions. In this chapter, we focus on the static network analysis methods using network representation learning which is also known as network embedding. Network embedding arises in many machine learning tasks assuming that there exist a small number of variabilities in the vertex representations which can capture the "semantics" of the original network structure. Most existing network embedding models, with shallow or deep architectures, learn vertex representations from the sampled vertex sequences such that the low-dimensional embeddings preserve the locality property and/or global reconstruction capability. The resultant representations, however, are difficult for model generalization due to the intrinsic sparsity of sampled sequences from the input network. As such, an ideal approach to address the problem is to generate vertex representations by learning a probability density function over the sampled sequences. However, in many cases, such a distribution in a low-dimensional manifold may not always have an analytic form. In this chapter we will introduce the NETRA model in Section 3.1 which learns the network representations with adversarially regularized autoencoders. The resultant vertex representations can well capture the network structure through jointly considering both locality-preserving and global reconstruction constraints. The joint inference is encapsulated in a generative adversarial training process to circumvent the requirement of an explicit prior distribution, and thus obtains better generalization performance.

3.1 Representation Learning with Adversarially Regularized Autoencoders

Static network analysis has been attracting many research interests with its enormous potential in mining useful information which benefits the downstream tasks such as link prediction, com-

munity detection and anomaly detection on social network [Tylenda et al., 2009], biological networks [Theocharidis et al., 2009] and language networks [Tang et al., 2015], to name a few.

To analyze network data, one fundamental problem is to learn a low-dimensional vector representation for each vertex, such that the network structure is preserved in the learned vector space [Perozzi et al., 2014b]. For this problem, there are two major challenges: (1) *preservation of complex structure property*. The objective of network embedding is to train a model to "fit" the training networks, that is, to preserve the structure property of networks [Perozzi et al., 2014b, Ribeiro et al., 2017]. However, the latent structure of the network is too complex to be portrayed by an explicit form of probability density which can capture both the local network neighborhood information and global network structure. (2) *sparsity of network sampling*. Current research on network embedding employs network sampling techniques, including random walk sampling, breadth-first search etc., to derive vertex sequences as training datasets. However the sampled data represent only a small proportion of all the vertex sequences. An alternative approach is to encode these discrete structures in a continuous code space [Wang et al., 2016]. Unfortunately, learning continues latent representations of discrete networks remains a challenging problem since in many cases, the prior distribution may not exist in a low dimensional manifold [Ribeiro et al., 2017].

Recent work on network embedding has shown fruitful progress in learning vertex representations of complex networks [Perozzi et al., 2014b, Ribeiro et al., 2017, Wang et al., 2016]. These representations employ nonlinear transformations to capture the "semantics" of the original networks. Most existing methods first employ a random walk technique to sample a bunch of vertex sequences from the input network, then feed a learning model with these sequences to infer the optimal low-dimensional vertex embeddings. However, the sampling strategy suffers from the data sparsity problem since the total amount of vertex sequences is usually very large in real networks and it is often intractable to enumerate all. Subsequently, learning on a sparse sample set tends to produce an overly complex model to explain the sampled dataset, which eventually causes overfitting. Though autoencoders are adopted to encode the inputs into continuous latent representations [Wang et al., 2016], regularizations are still desirable to force the learned representations remain on the latent manifold. Ideally we could generate the continuous vertex representations with a prior distribution. However, in many cases, it is difficult, if not impossible, to pre-define

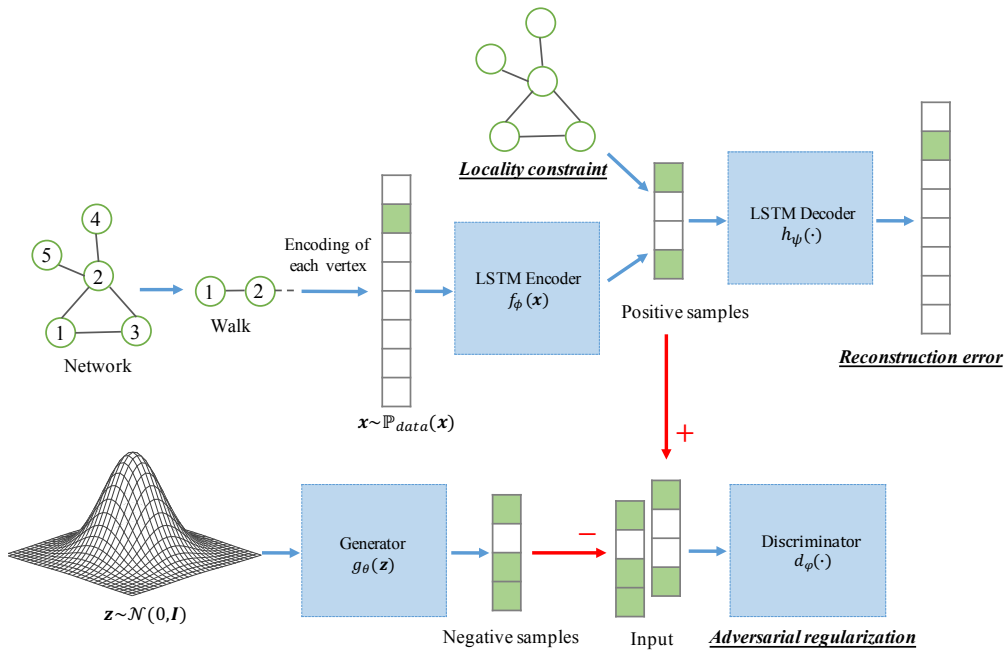


Figure 3.1: Illustration of the NETRA model

an explicit form of the prior distribution in a low-dimensional manifold. For example, Dai et al. [Dai et al., 2017] proposed to train a discriminator to distinguish samples generated from a fixed prior distribution and the input encoding, and thereby pushing the embedding distribution to match the fixed prior. While this gives more flexibility, it suffers from the mode-collapse problem [Kim et al., 2017]. Moreover, most network embedding models with deep architectures usually do not consider the order of the vertices in the sampled vertex sequences [Wang et al., 2016]. Thus, the information of proximity orders cannot be well considered.

To address the aforementioned challenges, in this study, we propose a novel model to learn the network representations with adversarially regularized autoendoers (NETRA). NETRA jointly minimizes network locality-preserving loss and the reconstruction error of autoencoder which utilizes a long short-term memory network (LSTM) as an encoder to map the input sequences into a fixed length representation. The joint embedding inference is encapsulated in a generative adversarial training process to circumvent the requirement of an explicit prior distribution. As visually depicted in Figure 3.1, our model employs a discrete LSTM autoencoder to learn continuous vertex representations with sampled sequences of vertices as inputs. In this model, besides minimizing the reconstruction error in the LSTM autoencoder, the locality-preserving loss at the hidden layer

is also minimized simultaneously. Meanwhile, the continuous space generator is also trained by constraining to agree in distribution with the encoder. The generative adversarial training can be regarded as a complementary regularizer to the network embedding process.

NETRA exhibits desirable properties that a network embedding model requires: 1) *structure property preservation*, NETRA leverages LSTM as an encoder to capture the neighborhood information among vertices in each sequence sampled from the network. Additionally, the model is also trained simultaneously with the locality-preserving constraint. 2) *generalization capability*, the generalization capability requires a network embedding model to generalize well on unseen vertex sequences which follow the same distribution as the population. The generative adversarial training process enables the proposed model to learn smoothly regularized representations without pre-defining an explicit density distribution which overcomes the sparsity issue from the input sequences of vertices, as described in Section 3.1.2. We present experimental results in Section 3.1.3 to show the embedding capability of NETRA on a variety of tasks, including network reconstruction, link prediction and vertex classification.

3.1.1 Preliminaries

3.1.1.1 Network Embedding

Network embedding approaches seek to learn representations that encode structural information about the network. These approaches learn a mapping that embeds vertices as points into a low-dimensional space. Given the encoded vertex set $\{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(n)}\}$, finding an embedding $f_\phi(\mathbf{x}^{(i)})$ of each $\mathbf{x}^{(i)}$ can be formalized as an optimization problem [Yu et al., 2013b, Weston et al., 2008]

$$\min_{\phi} \sum_{1 \leq i < j \leq n} L(f_\phi(\mathbf{x}^{(i)}), f_\phi(\mathbf{x}^{(j)}), \varphi_{ij}), \tag{3.1}$$

where $f_\phi(\mathbf{x}) \in \mathbb{R}^d$ is the embedding result for a given input $\mathbf{x}$. $L(\cdot)$ is the loss function between a pair of inputs. φ_{ij} is the weight between $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$.

We consider the Laplacian Eigenmaps (LE) that well fits the framework. LE enables the embedding to preserve the locality property of network structure. Formally, the embedding can be

obtained by minimizing the following objective function

$$\mathcal{L}_{LE}(\phi; \mathbf{x}) = \sum_{1 \leq i < j \leq n} \| f_\phi(\mathbf{x}^{(i)}) - f_\phi(\mathbf{x}^{(j)} \|^2 \varphi_{ij}. \tag{3.2}$$

3.1.1.2 Generative Adversarial Networks

The Generative Adversarial Networks (GANs) [Goodfellow et al., 2014] build an adversarial training platform for two players, namely *generator* $g_\theta(\cdot)$ and *discriminator* $d_w(\cdot)$, to play a minimax game.

$$\min_\theta \max_w \mathop{\mathbb{E}}_{\mathbf{x} \sim \mathbb{P}_{data}(\mathbf{x})} [\log d_w(\mathbf{x})] + \mathop{\mathbb{E}}_{\mathbf{z} \sim \mathbb{P}_g(\mathbf{z})} [\log (1 - d_w(g_\theta(\mathbf{z})))] \tag{3.3}$$

The *generator* $g_\theta(\cdot)$ tries to map the noise to the input space as closely as the true data, while the *discriminator* $d_w(\mathbf{x})$ represents the probability that $\mathbf{x}$ came from the data rather than the noise. It aims to distinguish *real* data distribution $\mathbb{P}_{data}(\mathbf{x})$ and *fake* sample distribution $\mathbb{P}_g(\mathbf{z})$, e.g. $\mathbf{z} \sim \mathcal{N}(0, \mathbb{I})$. Wasserstein GANs [Arjovsky et al., 2017] overcome unstable training problem by replacing Jensen-Shannon divergence with Earth-Mover (Wasserstein-1) distance, which considers solving the problem

$$\min_\theta \max_{w \in \mathcal{W}} \mathop{\mathbb{E}}_{\mathbf{x} \sim \mathbb{P}_{data}(\mathbf{x})} [d_w(\mathbf{x})] - \mathop{\mathbb{E}}_{\mathbf{z} \sim \mathbb{P}_g(\mathbf{z})} [d_w(g_\theta(\mathbf{z}))]. \tag{3.4}$$

The Lipschitz constraint $\mathcal{W}$ on discriminator has been kept by clipping the weights of the discriminator within a compact space [-c, c].

3.1.1.3 Autoencoder Neural Networks

An autoencoder neural network is trained to set the target values to be equal to the inputs. The network consists of two parts: an encoder $f_\phi(\cdot)$ that maps inputs ($\mathbf{x} \in \mathbb{R}^n$) to latent low-dimensional representations and a decoder $h_\psi(\cdot)$ that produces a reconstruction of the inputs. Specifically, given a data distribution $\mathbb{P}_{data}$, from which $\mathbf{x}$ is drawn from, i.e., $\mathbf{x} \sim \mathbb{P}_{data}(\mathbf{x})$, we want to learn representations $f_\phi(\mathbf{x})$ such that the output hypotheses $h_\psi(f_\phi(\mathbf{x}))$ is approximately equal to $\mathbf{x}$. The learning process is described simply as minimizing a cost function

$$\min \mathbb{E}_{\mathbf{x} \sim \mathbb{P}_{data}(\mathbf{x})} [\text{dist}(\mathbf{x}, h_\psi(f_\phi(\mathbf{x})))], \tag{3.5}$$

where dist$(\cdot)$ is some similarity metric in the data space. In practice, there are many options for the distance measure. For example, if we use ℓ_2 norm to measure the reconstruction error, then the objective function can be defined as $\mathcal{L}_{\text{AE}}(\phi, \psi; \mathbf{x}) = \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{data}(\mathbf{x})}\|\mathbf{x} - h_\psi(f_\phi(\mathbf{x}))\|^2$. Similarly the objective function for cross-entropy loss can be defined as,

$$-\mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{data}(\mathbf{x})}[\mathbf{x}\log h_\psi(f_\phi(\mathbf{x})) + (\mathbf{1}-\mathbf{x})\log(\mathbf{1} - h_\psi(f_\phi(\mathbf{x})))] \tag{3.6}$$

The choice of encoder $f_\phi(\cdot)$ and decoder $h_\psi(\cdot)$ may vary across different tasks. In this paper, we use LSTM autoencoders [Sutskever et al., 2014] which are capable of dealing with sequences as inputs.

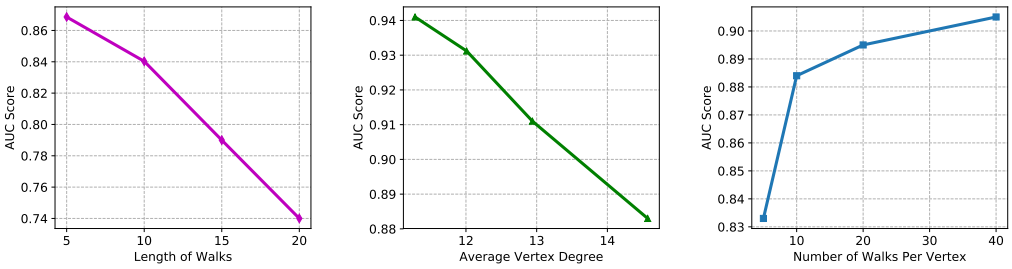
3.1.2 The NETRA Model

We consider the network embedding problem of finding a transformation that maps structural input data to lower-dimensional vertex representations. In this section, we present NETRA, a deep network embedding model using adversarially regularized autoencoders, to learn smoothly regularized vertex representations with sequences of vertices as inputs. The resultant representations can be used in the downstream tasks, such as link prediction, network reconstruction and multi-class classification.

3.1.2.1 Random Walk Generator

Given network $G(V, E)$, the random walk generator in DeepWalk [Perozzi et al., 2014b] is utilized to obtain truncated random walks (i.e. sequences of vertices) rooted on each vertex $v \in V$ in $G(V, E)$. A walk is sampled randomly from the neighbors of the last visited vertex until the preset maximum length is reached.

The random walk sampling technique has been widely adopted in network embedding researches [Grover and Leskovec, 2016, Perozzi et al., 2014b, Wang et al., 2016]. However, it suffers from the sparsity problem in network sampling. For each vertex in given network, if we assume that the average node degree is $\bar{d}$, the walk length is l and the number of samples is k, then



(a) Walk length (b) Vertex degree (c) Number of samples

Figure 3.2: Sparsity of network sampling.

the sampling fraction of walks can be calculated by

$$p_{frac} \propto \frac{|V| \times k}{|V| \times \bar{d}^l} = \frac{k}{\bar{d}^l} \times 100\%. \tag{3.7}$$

The effect of the sampling fraction is presented in Figure 3.2. In the example, DeepWalk is used to perform link prediction task on the *UCI Msg* network described in Section A.1. Figure 3.2(a) and Figure 3.2(b) show that if the walk length or the average vertex degree increases, the performance decreases dramatically[1]. According to Eq. (3.7), obviously, when l or $\bar{d}$ increases, the sampling fraction of walks is getting smaller. Thereby, the trained model is prone to overfitting because of the sparse inputs. On the contrary, if the number of samples k increases, the performance is getting better as shown in Figure 3.2(c). However, more sampled walks also call for more computing burden on model training. Therefore, it is desirable to develop effective models with better capabilities of generalization on sparsely sampled network walks.

3.1.2.2 Embedding with Adversarially regularized Autoencoders

In this section, we propose NETRA, a network embedding model with adversarially regularized autoencoders, to address the sparsity problem. Autoencoders are popularly used for data embedding, such as images and documents. It provides informative low dimensional representations of input data by mapping the them to the latent space. Unfortunately, if the encoder and decoder are

[1]In Figure 3.2(a), the window size of DeepWalk is set to be equal to the walk length. The reason is that, if the window size is set to a small value against a long walk length, it turns out to be equivalent to increase the samples per vertex with a short walk length. In Figure 3.2(b), we reduce the degree of the dataset by removing vertices with large degrees [**?**].

allowed too much capacity, the autoencoder can learn to perform the copying task without extracting useful information about the distribution of the data [Goodfellow et al., 2016]. We proposed to use a generative adversarial training process as a complementary regularizer. The process has two advantages. On one hand, the regularizer can guide the extraction of useful information about data [Goodfellow et al., 2016]. On the other hand, the generative adversarial training provides more robust discrete-space representation learning that can well address the overfitting problem on sparsely sampled walks [Makhzani et al., 2016]. Specifically, in NETRA, the discriminator updates by comparing the samples from the latent space of the autoencoder with the fake samples from the generator, as shown in Figure 3.1. The latent space of autoencoder provides optimal embedding for the vertices in the network with the simultaneous update of encoder and discriminator. In this study, we use the LSTM as the encoder and decoder networks [Sutskever et al., 2014] because it takes the order information of the sampled walks into consideration.

This joint architecture requires dedicated training objective for each part. The autoencoder can be trained individually by minimizing the negative log-likelihood of reconstruction, which is indicated by cross entropy loss in the implementation

$$\mathcal{L}_{\text{AE}}(\phi, \psi; \mathbf{x}) = -\mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{data}(\mathbf{x})}[\text{dist}(\mathbf{x}, h_\psi(f_\phi(\mathbf{x})))], \tag{3.8}$$

where $\text{dist}(\mathbf{x}, \mathbf{y}) = \mathbf{x}\log\mathbf{y} + (\mathbf{1}-\mathbf{x})\log(\mathbf{1}-\mathbf{y})$. Here $\mathbf{x}$ is the sampled batch from training data. $f_\phi(\mathbf{x})$ is embedded latent representation of $\mathbf{x}$, which is also the positive samples for discriminator, indicated by the arrow with "+" in Figure 3.1. ϕ and ψ are parameters of the encoder and decoder functions, respectively. In the training iteration of autoencoder, not only the encoder and decoder are updated, the locality-preserving loss (Eq. (3.2)) is jointly minimized.

As depicted in Figure 3.1, NETRA minimizes the distributions between the learned representations from the encoder function $f_\phi(\mathbf{x}) \sim \mathbb{P}_\phi(\mathbf{x})$, and the representations from the continuous generator model $g_\theta(\mathbf{z}) \sim \mathbb{P}_\theta(\mathbf{z})$. The dual form of the Earth Mover distance between $\mathbb{P}_\phi(\mathbf{x})$ and $\mathbb{P}_\theta(\mathbf{z})$ can be described as follows [Arjovsky et al., 2017]

$$W(\mathbb{P}_\phi(\mathbf{x}), \mathbb{P}_\theta(\mathbf{z})) = \sup_{\|d(\cdot)\|_{L\leq 1}} \mathbb{E}_{\mathbf{y}\sim\mathbb{P}_\phi(\mathbf{x})}[d(\mathbf{y})] - \mathbb{E}_{\mathbf{y}\sim\mathbb{P}_\theta(\mathbf{z})}[d(\mathbf{y})] \tag{3.9}$$

where $\|d(\cdot)\|_{L\leq 1}$ is the Lipschitz continuity constraint (with Lipschitz constant 1). If we have a

family of functions $\{d_w(\cdot)\}_{w\in\mathcal{W}}$ that are all K-Lipschitz for some K, then we have

$$W(\mathbb{P}_\phi(\mathbf{x}), \mathbb{P}_\theta(\mathbf{z})) \propto \max_{w\in\mathcal{W}} \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathbb{P}_{data}(\mathbf{x})}[d_w(f_\phi(\mathbf{x}))] - \mathop{\mathbb{E}}_{\mathbf{z}\sim\mathbb{P}_g(\mathbf{z})}[d_w(g_\theta(\mathbf{z}))] \tag{3.10}$$

We can separate the training of generator and discriminator. As for the generator, the cost function can be defined as,

$$\mathcal{L}_{\text{GEN}}(\theta; \mathbf{x}, \mathbf{z}) = \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{data}(\mathbf{x})}[d_w(f_\phi(\mathbf{x}))] - \mathbb{E}_{\mathbf{z}\sim\mathbb{P}_g(\mathbf{z})}[d_w(g_\theta(\mathbf{z}))] \tag{3.11}$$

and the cost function for discriminator is,

$$\mathcal{L}_{\text{DIS}}(w; \mathbf{x}, \mathbf{z}) = -\mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{data}(\mathbf{x})}[d_w(f_\phi(\mathbf{x}))] + \mathbb{E}_{\mathbf{z}\sim\mathbb{P}_g(\mathbf{z})}[d_w(g_\theta(\mathbf{z}))] \tag{3.12}$$

NETRA learns smooth representations by jointly minimizing the autoencoder reconstruction error and the locality-preserving loss in an adversarial training process. Specifically, we consider solving the joint optimization problem with objective function

$$\mathcal{L}_{\text{NETRA}}(\phi, \psi, \theta, w) = \mathcal{L}_{\text{AE}}(\phi, \psi; \mathbf{x}) + \lambda_1 \mathcal{L}_{\text{LE}}(\phi; \mathbf{x}) + \lambda_2 W(\mathbb{P}_\phi(\mathbf{x}), \mathbb{P}_\theta(\mathbf{z})) \tag{3.13}$$

Theorem 1. *Let $\mathbb{P}_\phi(\mathbf{x})$ be any distribution. Let $\mathbb{P}_\theta(\mathbf{z})$ be the distribution of $g_\theta(\mathbf{z})$ with $\mathbf{z}$ being a sample drawn from distribution $\mathbb{P}_g(\mathbf{z})$ and $g_\theta(\cdot)$ being a function satisfying the local Lipschitz constants $\mathbb{E}_{\mathbf{z}\sim\mathbb{P}_g(\mathbf{z})}[L(\theta, \mathbf{z})] < +\infty$. Then we have*

$$\begin{aligned}
\nabla_\theta \mathcal{L}_{\text{NETRA}} &= -\lambda_2 \nabla_\theta \mathbb{E}_{\boldsymbol{z}\sim\mathbb{P}_g(\mathbf{z})}[d_w(g_\theta(\boldsymbol{z}))] && (3.14)\\
\nabla_w \mathcal{L}_{\text{NETRA}} &= -\lambda_2 \nabla_w \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_{data}(\boldsymbol{x})}[d_w(f_\phi(\boldsymbol{x}))] + \lambda_2 \nabla_w \mathbb{E}_{\boldsymbol{z}\sim\mathbb{P}_g(\mathbf{z})}[d_w(g_\theta(\boldsymbol{z}))] && (3.15)\\
\nabla_\phi \mathcal{L}_{\text{NETRA}} &= \lambda_1 \nabla_\phi \sum_{1\le i<j\le n} \|f_\phi(\mathbf{x}^{(i)}) - f_\phi(\mathbf{x}^{(j)})\|^2 \varphi_{ij} \\
&\quad -\nabla_\phi \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{data}(\mathbf{x})}[\boldsymbol{dist}(\mathbf{x}, h_\psi(f_\phi(\mathbf{x})))] + \lambda_2 \nabla_\phi \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_{data}(\boldsymbol{x})}[d_w(f_\phi(\boldsymbol{x}))] && (3.16)\\
\nabla_\psi \mathcal{L}_{\text{NETRA}} &= -\nabla_\psi \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{data}(\mathbf{x})}[\boldsymbol{dist}(\mathbf{x}, h_\psi(f_\phi(\mathbf{x})))] && (3.17)
\end{aligned}$$

Proof. Let $\mathcal{X} \subseteq \mathbb{R}^n$ be a compact set, and

$$\begin{aligned}
V(\tilde{d}, \theta) &= \mathbb{E}_{\mathbf{y}\sim\mathbb{P}_\phi(\mathbf{x})}[\tilde{d}(\mathbf{y})] - \mathbb{E}_{\mathbf{y}\sim\mathbb{P}_\theta(\mathbf{z})}[\tilde{d}(\mathbf{y})] \\
&= \mathbb{E}_{\mathbf{y}\sim\mathbb{P}_\phi(\mathbf{x})}[\tilde{d}(\mathbf{y})] - \mathbb{E}_{\mathbf{z}\sim\mathbb{P}_g(\mathbf{z})}[\tilde{d}(g_\theta(\mathbf{z}))] && (3.18)
\end{aligned}$$

where $\tilde{d}$ lies in $\mathcal{D} = \{\tilde{d} : \mathcal{X} \to \mathbb{R}, \tilde{d} \text{ is continuous and bounded}, \|\tilde{d}\| \leq 1\}$. Since $\mathcal{X}$ is compact, we know by the Kantorovich-Rubinstein duality [Arjovsky et al., 2017] that there exists a $d \in \mathcal{D}$ that attains the value

$$W(\mathbb{P}_\phi(\mathbf{x}), \mathbb{P}_\theta(\mathbf{z})) = \sup_{\tilde{d} \in \mathcal{D}} V(\tilde{d}, \theta) = V(d, \theta) \tag{3.19}$$

and $D^*(\theta) = \{d \in \mathcal{D} : V(d, \theta) = W(\mathbb{P}_\phi(\mathbf{x}), \mathbb{P}_\theta(\mathbf{z}))\}$ is non-empty. According to the envelope theorem [Milgrom and Segal, 2002], we have

$$\nabla_\theta W(\mathbb{P}_\phi(\mathbf{x}), \mathbb{P}_\theta(\mathbf{z})) = \nabla_\theta V(d, \theta) \tag{3.20}$$

for any $d \in D^*(\theta)$. Then we get

$$\begin{aligned} \nabla_\theta W(\mathbb{P}_\phi(\mathbf{x}), \mathbb{P}_\theta(\mathbf{z})) &= \nabla_\theta V(d, \theta) \\ &= \nabla_\theta \mathbb{E}_{\mathbf{y} \sim \mathbb{P}_\phi(\mathbf{x})}[d(\mathbf{y})] - \mathbb{E}_{\mathbf{z} \sim \mathbb{P}_g(\mathbf{z})}[d(g_\theta(\mathbf{z}))] \\ &= -\nabla_\theta \mathbb{E}_{\mathbf{z} \sim \mathbb{P}_g(\mathbf{z})}[d_w(g_\theta(\mathbf{z}))] \end{aligned} \tag{3.21}$$

Therefore, we have $\nabla_\theta \mathcal{L}_{\text{NETRA}} = -\lambda_2 \nabla_\theta \mathbb{E}_{\mathbf{z} \sim \mathbb{P}_g(\mathbf{z})}[d_w(g_\theta(\mathbf{z}))]$.

Eq. (3.15)-(3.17) are straightforward applications of the derivative definition. □

We now have all the derivatives needed. To train the model, we use a block coordinate descent to alternate between optimizing different parts of the model: (1) locality-preserving loss and autoencoder reconstruction error (update ϕ and ψ), (2) the discriminator in the adversarial training process (update w), and (3) the generator (update θ). Pseudocode of the full approach is given in Algorithm 1.

The training process of NETRA consists of the following steps: Firstly, given a network $G(V, E)$, we run random walk generator acquiring random walks of length l. Then, one hot representation $\mathbf{x}^{(i)}$ of each vertex is taken as input to LSTM cells. We pass the random walks through encoding layers and obtain the vector representations of vertices. After the decoder network, the vertex representations will be transformed back into n dimensions. Cross-entropy loss is calculated between the inputs and outputs by minimizing the reconstruction error in autoencoder operation. Meanwhile, locality-preserving constraint ensures that the adjacent vertices are in close proximity (Step 2-7 in Algorithm 1). The latent representation of encoder and the output of generator will be

Algorithm 1 NETRA Model Training

Require: the walks generated from input graph, maximum training epoch n_{epoch}, the number of discriminator training per generator iteration n_D.

for $epoch = 0; epoch < n_{epoch}$ **do**
  ▷ *Training autoencoder with LE constraint*
  Minimizing $\mathcal{L}_{\text{LE}}(\phi; \mathbf{x})$ **with autoencoder** $\mathcal{L}_{\text{AE}}(\phi, \psi; \mathbf{x})$
  Sample $\{\mathbf{x}^{(i)}\}_{i=1}^{B} \sim \mathbb{P}_{data}(\mathbf{x})$ a batch from the walks
  Compute latent representation $f_\phi(\mathbf{x}^{(i)})$
  Compute reconstruction output $h_\psi(f_\phi(\mathbf{x}^{(i)}))$
  Compute $\mathcal{L}_{\text{AE}}(\phi, \psi)$ and $\mathcal{L}_{\text{LE}}(\phi)$ using Eq. (3.8) and Eq. (3.2)
  Back-propagate loss and update ϕ and ψ using Eq. (3.16)-(3.17)
  ▷ *Training discriminator*
  for $n = 0, n < n_D$ **do**
    Sample $\{\mathbf{x}^{(i)}\}_{i=1}^{B} \sim \mathbb{P}_{data}(\mathbf{x})$ a batch from the walks
    Sample $\{\mathbf{z}^{(i)}\}_{i=1}^{B} \sim \mathbb{P}_{g}(\mathbf{z})$ a batch from the noise
    Compute representations $f_\phi(\mathbf{x}^{(i)})$ and $g_\theta(z^{(i)})$
    Compute $\mathcal{L}_{\text{DIS}}(w)$ using Eq. (3.12)
    Back-propagate loss and update w using Eq. (3.15)
    clip the weight w within $[-c, c]$
  end for
  ▷ *Training generator*
  Sample $\{\mathbf{z}^{(i)}\}_{i=1}^{B} \sim \mathbb{P}_{g}(\mathbf{z})$ a batch from the noise
  Compute the representation $g_\theta(\mathbf{z}^{(i)})$
  Compute $\mathcal{L}_{\text{GEN}}(\theta)$ using Eq. (3.11)
  Back-propagate loss and update θ using Eq. (3.14)
end for

fed into discriminator to get adversarial loss (Step 10-17). Additionally, the generator transforms Gaussian noise into the latent space as closely as the true data, by passing through multilayer perceptron (Step 20-23). After the training of NETRA, we obtain the vertex representations $f_\phi(\mathbf{x})$ of the network by passing the input walks through the encoder function.

Optimality Analysis. NETRA, as illustrated in Figure 3.1, can be interpreted as minimizing the divergence between two distributions, namely $\mathbb{P}_\phi(\mathbf{x})$ and $\mathbb{P}_\theta(\mathbf{z})$. We provide the following proposition which shows that under our parameter settings, if the Wasserstein distance converges, the encoder distribution $f_\phi(\mathbf{x}) \sim \mathbb{P}_\phi(\mathbf{x})$ converges to the generator distribution $g_\theta(\mathbf{z}) \sim \mathbb{P}_\theta(\mathbf{z})$.

Theorem 2. *Let $\mathbb{P}$ be a distribution on a compact set $\mathcal{X}$, and $(\mathbb{P}_n)_{\in\mathbb{N}}$ be a sequence of distributions on $\mathcal{X}$. Considering $W(\mathbb{P}_n, \mathbb{P}) \to 0$ as $n \to \infty$, the following statements are equivalent:*

1. *$\mathbb{P}_n \overset{\mathcal{D}}{\rightsquigarrow} \mathbb{P}$ where $\overset{\mathcal{D}}{\rightsquigarrow}$ represents convergence in distribution for random variables.*
2. *$\mathbb{E}_{\mathbf{x}\sim\mathbb{P}_n}[F(\mathbf{x})] \to \mathbb{E}_{\mathbf{x}\sim\mathbb{P}}[F(\mathbf{x})]$, where $F(\mathbf{x}) = \Pi_{i=1}^n \mathbf{x}_i^{p_i}, \mathbf{x} \in \mathbb{R}^n, \sum_{i=1}^n p_i = k, k > 1, k \in \mathbb{N}$.*

Proof. (1) As shown in [Villani, 2008], $\mathbb{P}_n$ converges to $\mathbb{P}$ is equivalent to $W(\mathbb{P}_n, \mathbb{P}) \to 0$.

(2) According to the *Portmanteau Theorem* [Villani, 2008], $\mathbb{E}_{\mathbf{x}\sim\mathbb{P}_n}[F(\mathbf{x})] \to \mathbb{E}_{\mathbf{x}\sim\mathbb{P}}[F(\mathbf{x})]$ holds if $F : \mathbb{R}^n \to \mathbb{R}$ is a bounded continuous function. Our encoder $f_\phi(\cdot)$ is bounded as the inputs are normalized to lie on the unit sphere, and our generator $g_\theta(\cdot)$ is also bounded to lie in $(-1, 1)^n$ by *tanh* function. Therefore, $F(\mathbf{x}) = \Pi_{i=1}^n \mathbf{x}_i^{p_i}$ is a bounded continuous function for all $p_i > 0$, and

$$\mathbb{E}_{\mathbf{x}\sim\mathbb{P}_n}[\Pi_{i=1}^n \mathbf{x}_i^{p_i}] \to \mathbb{E}_{\mathbf{x}\sim\mathbb{P}}[\Pi_{i=1}^n \mathbf{x}_i^{p_i}] \tag{3.22}$$

such that $\sum_{i=1}^n p_i = k, \forall k > 1, k \in \mathbb{N}$. □

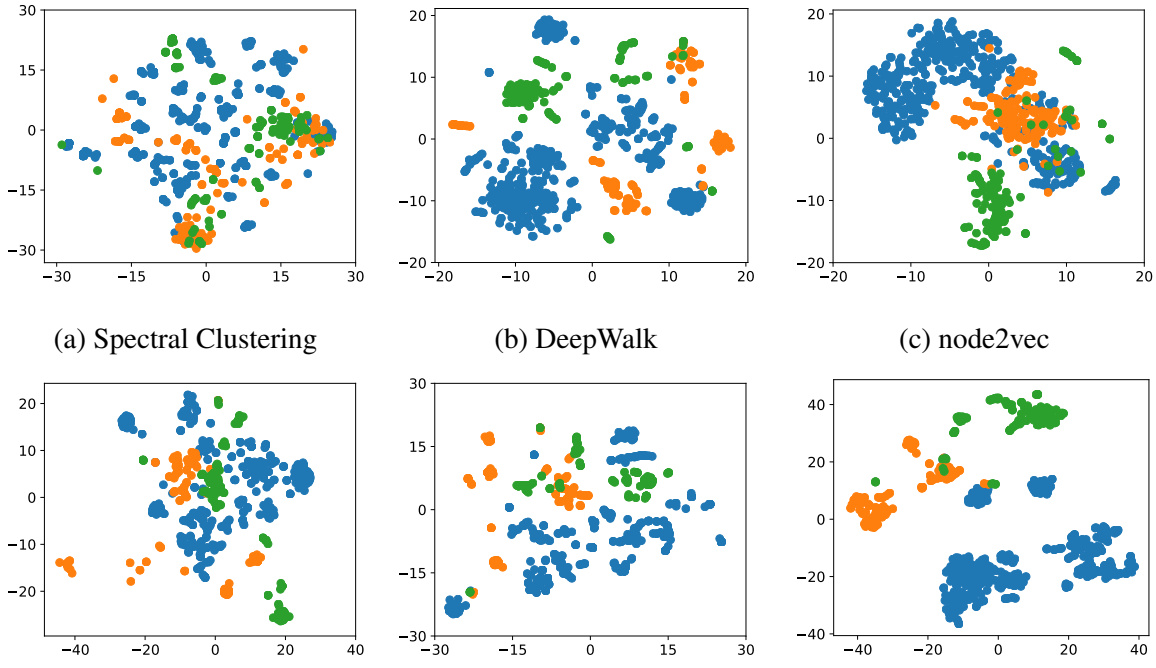
Computational Analysis. Given a network $G(V, E)$, where $|V| = n, |E| = m$, according to the definition in Eq. (3.2), the overall complexity of Laplacian Eigenmaps embedding is $\mathcal{O}(n^2)$. In our implementation, we only consider the vertex pairs $(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$ that have edges between them, thus the size of the sampled pairs is $\mathcal{O}(m)$, which is much smaller than $O(n^2)$ because real networks are sparse in real settings. The computational complexity of learning LSTM autoencoders is proportional to the number of parameters $|\phi|$ and $|\psi|$ in each iteration. Therefore, the learning

computational complexity for LSTM autoencoders is $\mathcal{O}(n_{epoch} \times (|\phi| + |\psi|))$. Similarly, for the generator and discriminator, each invocation of back-propagation is typically linear in the number of parameters $\mathcal{O}(|\theta|)$ and $\mathcal{O}(|w|)$. Thus the computational complexity for generator and discriminator is $\mathcal{O}(n_{epoch} \times (n_D \times |w| + |\theta|))$. It is basically quadratic if the input and hidden layers are of roughly the same size. However, if we set the size of embedding layers much less than that of the inputs, the time complexity reduces to $\mathcal{O}(n)$.

3.1.3 Evaluation

We evaluate the performance of our model with extensive experiments on tasks including network reconstruction, link prediction and vertex classification. The experiments are conducted on a variety of networks from different domains including *UCI Msg*, *JDK*, *Blogcatalog*, *DBLP*, *Wikipedia* and *PPI*. The detailed description of each dataset can be found in Section A.1.

To evaluate the performance of our network embedding model, the baselines used in this study includes Spectral Clustering (SC) [Tang and Liu, 2011], DeepWalk [Perozzi et al., 2014b], node2vec [Grover and Leskovec, 2016], SDNE [Wang et al., 2016] and Adversarial Network Embedding (ANE) [Dai et al., 2017], as described in Section A.2. For fair comparison, we run each algorithm to generate 300 dimensional vertex representations on different datasets, unless noted otherwise. The number of walks per vertex in DeepWalk and node2vec is set to 10 with walk length 30, which is the same as the random walk generation step of NETRA. The window size of DeepWalk and node2vec is optimized to 10. node2vec is optimized with grid search over its return and in-out parameters $(p, q) \in \{0.25, 0.50, 1, 2, 4\}$. For SDNE, we utilize the default parameter setting as described in [Wang et al., 2016]. For NETRA, the gradient clipping is performed in every training iteration to avoid the gradient explosion, and we use stochastic gradient descent as the optimizer of autoencoder networks. The multilayer perceptron (MLP) is used in the generator and discriminator. The evaluation of different algorithms is based on applying the embeddings they learned to the downstream tasks, such as link prediction, network reconstruction, and multi-label classification as will be illustrated in the subsequent sections.



(a) Spectral Clustering (b) DeepWalk (c) node2vec

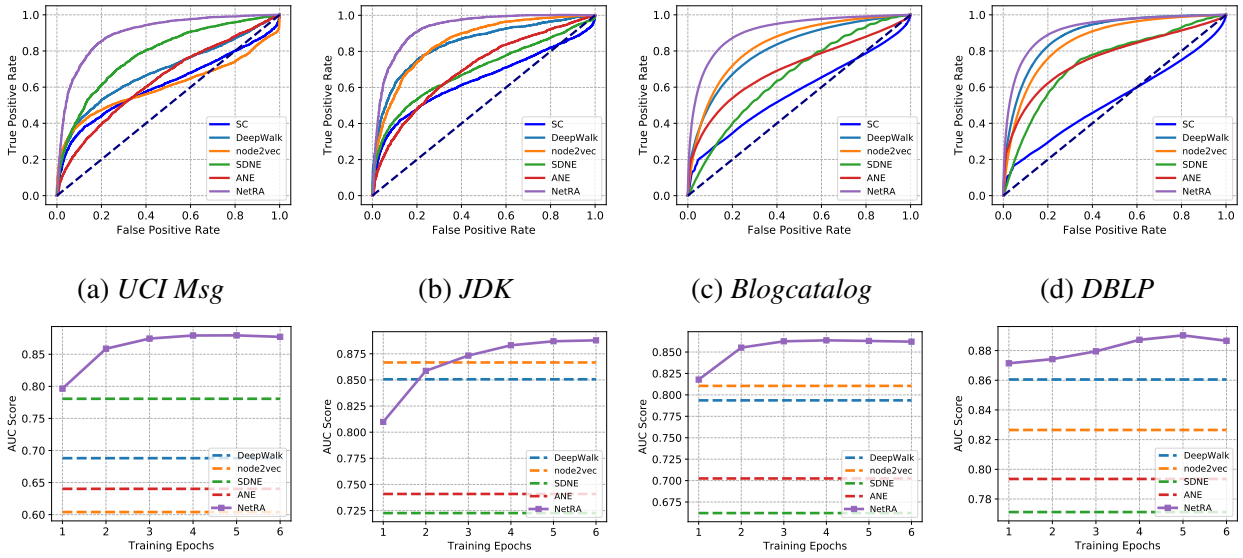
(d) SDNE (e) ANE (f) NETRA

Figure 3.3: Visualization results of the compared methods on *JDK* dataset

3.1.3.1 Visualization

In order to demonstrate how well key properties of network structure are captured by the network embedding models, we visualize the embeddings of each compared method. We run different embedding algorithms to obtain low dimensional representations of each vertex and map vertex vectors onto a two dimensional space using t-SNE [Maaten and Hinton, 2008]. With vertex colored by its label, we perform the visualization task on *JDK* network, as shown in Figure 3.3.

As observed in Figure 3.3, three classes are presented: red points for *org.omg*, green points for *org.w3c* and blue points for *java.beans*. It can be seen that the eigenvector-based method Spectral Clustering cannot effectively identify different classes. Other baselines can detect the classes to varying extents. NETRA performs best as it can separate these three classes with large boundaries, except for a small overlap between green and red vertices.



(a) *UCI Msg* (b) *JDK* (c) *Blogcatalog* (d) *DBLP*

(e) *UCI Msg* (f) *JDK* (g) *Blogcatalog* (h) *DBLP*

Figure 3.4: Link prediction performance comparison

3.1.3.2 Link Prediction

The objective of link prediction task is to infer missing edges given a network with a certain fraction of edges removed. We randomly remove 50% of edges from the network, which serve as positive samples, and select an equal number of vertex pairs without linkage between them as negative samples. With vertex representation learned by network embedding algorithms, we obtain the edge feature from the ℓ_2 norm of two vertex vectors, and use it directly to predict missing edges. Because our focus is network embedding model, this simple experimental setup can evaluate the performance based on the assumption that the representations of two connected vertices should be closer in the Euclidean space. We use the area under curve (AUC) score for evaluation on link prediction task. The results are shown in Table 3.1.

Obviously, we observe that NETRA outperforms the baseline algorithms across all datasets by a large margin. It can be seen that NETRA achieves 3% to 32% improvement based on the AUC score on the four datasets. By comparing NETRA, node2vec and DeepWalk, which all use random walks as inputs, we can see the effectiveness of generative adversarial regularization for improving the generalization performance in NETRA model. With same random walk sequences, NETRA

Table 3.1: AUC score comparison on link prediction task

Methods	UCI Msg	JDK	Blogcatalog	DBLP
SC	0.6128	0.6686	0.6014	0.5740
DeepWalk	0.6880	0.8506	0.7936	0.8605
node2vec	0.6040	0.8667	0.8105	0.8265
SDNE	0.7806	0.7226	0.6621	0.7712
ANE	0.6402	0.7409	0.7025	0.7935
NETRA	**0.8879**	**0.8913**	**0.8627**	**0.8902**

can overcome the sparsity issue from the sampled sequences of vertices.

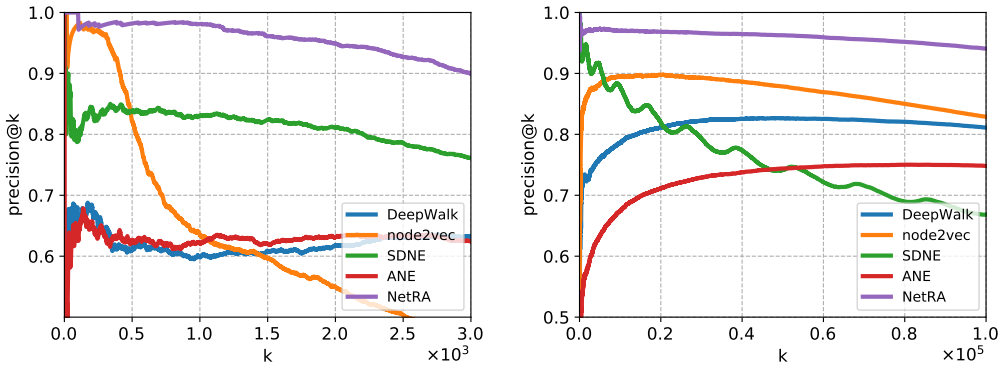
We also plot the ROC curve of these four datasets, as shown in Figure 3.4(a)-(d). The ROC curve of NETRA dominates other approaches and is very close to the $(0, 1)$ point. We train the NETRA model with different epochs for different datasets and embed the vertices to get representations after each training epoch. The results are shown in Figure 3.4(e)-(h). Generally, we can observe that NETRA converges pretty fast with high AUC score almost after the first epoch. When comparing with Deepwalk, node2vec, SDNE and ANE, we can clearly see the better performance of NETRA on these datasets.

3.1.3.3 Network Reconstruction

Network embeddings are considered as effective representations of the original network. The vertex representations learned by networking embedding maintain the edge information for network reconstruction. We randomly select vertex pairs as edge candidates and calculate the Euclidean distance between the vertices. We use the *precision*@k, the fraction of correct predictions in the top k predictions, for evaluation.

$$\textit{precision}@k = \frac{1}{k} \times |E_{\textit{pred}}(1:k) \cap E_{\textit{obs}}|, \tag{3.23}$$

where $E_{\textit{pred}}(1:k)$ represents the top k predictions and $E_{\textit{obs}}$ represents observed edges in original network. In the evaluation, the UCI message and Blogcatalog datasets have been utilized to



(a) UCI (b) BLOG

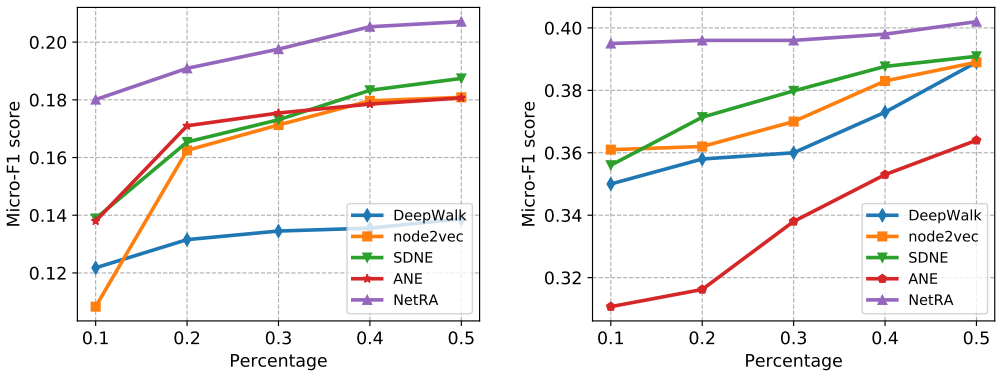
Figure 3.5: Network reconstruction results on *UCI Msg* and *Blogcatalog* dataset

illustrate the performance of NETRA, with results shown in Figure 3.5.

As it can be seen from the *precision@k* curves, the NETRA model achieves higher precision in the network reconstruction task. The total number of edge candidates selected in this task is $8k$ for UCI message and $300k$ for *Blogcatalog*. The reconstruction given by NETRA is very accurate in predicting most positive samples (results on JDK and DBLP datasets show similar trends which haven't been included here). DeepWalk and node2vec can give reasonable reconstruction but the results are worse than NETRA for most k's. By learning smoothly regularized vertex representations using generative adversarial training process [Goodfellow et al., 2014], our model well integrates the locality-preserving and global reconstruction constraints to learn embeddings that capture the "semantic" information.

3.1.3.4 Vertex Classification

The task of predicting vertex labels with the learned network representations is widely used in recent studies [Perozzi et al., 2014b, Grover and Leskovec, 2016, Wang et al., 2016]. An effective network embedding algorithm should capture network topology and extract most useful features for downstream machine learning tasks. In this section, we use vertex features as input to a one-vs-rest logistic regression using the LIBLINEAR [Fan et al., 2008] package to train the classifiers. For the *Wikipedia* and *PPI* datasets, we randomly sample 10% to 50% vertex labels as the training set and use the remaining vertices as the test set. We report *Micro-F1* [Wang et al., 2016] as evaluation



(a) PPI (b) WIKI

Figure 3.6: Vertex classification results on *PPI* and *Wikipedia* dataset

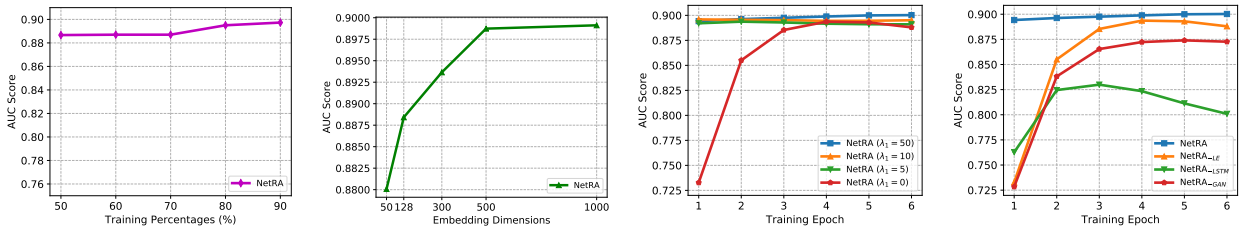
metrics. Each result is averaged by five runs, as shown in Figure 3.6.

It is evident from the figure that NETRA outperforms the state-of-the-art embedding algorithms on multi-label classification task. In the *PPI* dataset, NETRA achieves higher *Micro-F1* scores than the baseline models by over 10% in all experiment settings. In the *Wikipedia* dataset, NETRA model performs better even with lower percentage training set. This well illustrates the good generalization performance when the training set is sparse. The vertex classification task shows that, with adversarially regularized LSTM autoencoders, the neighborhood information can be well captured by the low dimensional representations.

3.1.3.5 Parameter Sensitivity

In this section, we investigate the parameter sensitivity in NETRA for link prediction. We study how the training set size, embedding dimension and locality-preserving constraint parameter λ_1 will affect the performance of link prediction. Also by changing the architecture of the NETRA model, we can investigate roles of different components in NETRA. Note that similar observations can be made on vertex classification and network reconstruction tasks.

In Figure 3.7(a), we vary the training percentage of edges in the UCI message network. As it can be seen, the performance increases as the training ratio increases. Comparing with other algorithms, NETRA can capture the network topology even with a small proportion of edges for



(a) Edge percentage (b) Dimension (c) λ_1 (d) NETRA variants

Figure 3.7: Parameter sensitivity analysis

training, which demonstrates the generalization capability of the NETRA model. In Figure 3.7(b), we vary the embedding dimension from 50 to 1000. The prediction performance gets saturated as the dimension increases. Considering that the embedding dimension is related to the parameter volume in NETRA, there exists a trade off between the performance and the efficiency during model training.

The parameter λ_1 is defined by the relative strength between locality-preserving constraint and autoencoder constraint. The higher the λ_1, the larger the gradient comes from the locality-preserving constraint. As observed from the Figure 3.7(c), a higher λ_1 enhances the link prediction performance on the UCI message network, indicating the important role of local proximity.

We also include three variants of NETRA to demonstrate the importance of individual components in NETRA, including NETRA$_{-LE}$, NETRA$_{-LSTM}$, and NETRA$_{-GAN}$. NETRA$_{-LE}$ and NETRA$_{-GAN}$ remove the locality-preserving constraint $\mathcal{L}_{\text{LE}}$ and adversarial regularization, respectively. As for NETRA$_{-LSTM}$, we replace LSTM with multilayer perceptron. It's evident from Figure 3.7(d) that LSTM autoencoder, locality-preserving constraint, and adversarial regularization play important roles in NETRA model. The overfitting becomes obvious in the training of NETRA$_{-LSTM}$ and NETRA$_{-GAN}$.

3.2 Summary

This chapter describes one of the static network analysis methods NETRA, which is a deep network embedding model for encoding each vertex in a network as a low-dimensional vector representa-

tion with adversarially regularized autoencoders. NETRA demonstrates the ability of generative adversarial training process in extracting informative representations. This model has better generalization capability, without requiring an explicit prior density distribution for the latent representations. Specifically, NETRA leverages LSTM autoencoders that take the sampled sequences of vertices as input to learn smooth vertex representations regularized by locality-preserving constraint and generative adversarial training process. The resultant representations are robust to the sparse vertex sequences sampled from the network. Empirically, the learned representations has been evaluated with a variety of network datasets on different tasks such as network reconstruction, link prediction and vertex classification. The results show substantial improvement over the state-of-the-art network embedding competitors.

CHAPTER 4

Static Network Analysis: Applications

Static network analysis has been extensively used in a wide range of applications such as propagation modeling, user behavior analysis, link prediction and community detection. In this chapter, we will introduce one of the community detection applications which is to detect competitive advertiser communities in a sponsored search market. The problem of measuring the intensity of competition among advertisers is increasingly gaining prominence today. Usually, search providers want to monitor the advertiser communities that share common bidding keywords, so that they can intervene when competition slackens. However, not much work has been conducted in identifying advertiser communities and understanding competition within these communities. Section 4.1 introduces a novel approach to detect competitive communities in a weighted bi-partite network formed by advertisers and their bidding keywords. The proposed approach is based on an advertiser vertex metric called *intensity score*, which takes the following two factors into consideration: the competitors that bid on the same keywords, and the advertisers' consumption proportion within the community. The community detection algorithm *Max-Intensity* is designed to detect communities which have the maximum intensity score.

4.1 Detecting Competitive Advertiser Communities

Search providers often divide the entire sponsored search market into different areas of business interests. Each such area, like healthcare, education, food and nutrition, etc. is formally known as a sector. Each sector has a wide spectrum of sponsored keywords that different advertisers bid on through keyword auctions. However, advertisers are eventually charged only when their sponsored ads are clicked by a user [Graepel et al., 2010]. Typically, advertisers open their accounts with the

search provider(s) and bid for a set of keywords which they consider to be relevant to their products [Perlich et al., 2012]. Different advertisers, bidding on the same set of keywords, compete against each other for their ad slot in the search results. Thus, the search engine providers often ask themselves *"what is the best way to measure the competition among different advertisers in each sector?"*. This is a very important question from the search providers' perspective, as understanding the intensity of competition among different advertisers can help them monitor a sector better, and if necessary, implement changes in their current policy to increase their revenue.

In traditional retail markets, market concentration measures which are based on firms' profits or market shares [Boone, 2008, Hirschman, 1964, Horvath, 1970] are used to estimate the competition. We can extend this notion to the sponsored search market as well, where we can view the fraction of the total "user clicks" that an advertiser gets on its sponsored ads as its corresponding market share in the sector. The fundamental principles of economic theory suggest that as market competition rises, the revenue for search providers should also increase [Shapiro, 2010]. This is easily verifiable from our results in Section 4.1.1. However, analyzing market competition at the sector level over time may not always give adequate insights. For example, if the competition for a sector remains stagnant or increases over time, it does not necessarily mean that the competition among advertisers for different keywords will follow a similar trend. Even in such cases, there will be pockets of advertisers, fiercely competing against each other for certain keywords, thereby contributing towards a higher search engine revenue. Likewise, a low competition for an entire sector does not indicate that all keywords within that sector are experiencing high competition. There can be several groups of keywords which are not fairing well in the market and are hardly sought out by the advertisers. Thus, our primary motivation is to detect small meaningful communities of advertisers, competing against each other, *within* a sector. These communities offer an appropriate microscopic view that will allow us to extract relevant insights like finding core and fringe competitors of advertisers, studying how the competition evolves, etc.

Traditional community detection algorithms [Newman, 2006b, Rosvall and Bergstrom, 2007, Newman, 2006a, Blondel et al., 2008], running on network graphs, allow us to discover groups of tightly connected vertices and their inter-relationships. In order to run these community detection methods, we modeled our sponsored search market as a bi-partite advertiser-keyword network, as

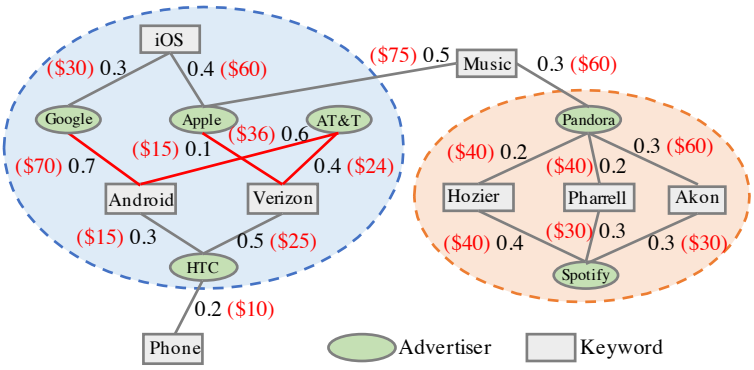


Figure 4.1: Weighted advertiser-keyword bipartite graph

shown in Figure 4.1 (detailed description can be found in Section 4.1.2). However, recent studies in community detection mainly focus either on improving the existing modularity-based methods [Kloster and Gleich, 2014, Duan et al., 2014], or proposing new metrics to better estimate the community structure [Chakraborty et al., 2014]. But, for our advertiser-keyword network, we need to detect communities based on their internal competition with each other. Therefore, in this study, we first introduce a novel scoring function called *intensity score* to measure the competitiveness of an advertiser. Thereafter, we propose a community detection algorithm *Max-Intensity* which detects highly competitive communities by maximizing the total intensity score within each community.

4.1.1 Preliminaries

Market concentration measures are commonly used in retail markets to estimate the competition among the stakeholders. One such measure that is widely regarded by economists as an excellent indicator for market competition is the Herfindahl-Hirschman index (HHI) [Hirschman, 1964]. HHI for a market having N firms is calculated as,

$$\text{HHI} = \sum_{i=1}^{N} s_i^2 \tag{4.1}$$

where s_i is the market share of firm i in the market, and N is the number of firms. Thus, in a market with two firms each having 50% market share, the HHI will be $0.5^2 + 0.5^2 = 0.5$.

Another alternative market concentration measure commonly used for estimating competition

is the comprehensive concentration index (CCI) [Horvath, 1970]. CCI for a market having N firms is calculated as,

$$\text{CCI} = s_1 + \sum_{i=2}^{N} s_i^2 \times (2 - s_i) \tag{4.2}$$

where s_1 is the market share of the largest competitor. Thus, in a market with two competitors having shares of 40% and 60%, the CCI will be $0.6 + 0.4^2 \times (2 - 0.4) = 0.856$. CCI puts greater weight on the share of the largest firm as compared to HHI. But typically, a low value of HHI or CCI indicates high market competition, whereas a high HHI or CCI would practically indicate a monopoly. We use both HHI and CCI metrics to validate our results.

In order to apply HHI and CCI in the context of a sponsored search marketplace, we can consider "user clicks" as sales for an advertiser and the fraction of total clicks garnered by the advertiser on its sponsored ads as its market share. With these considerations, we calculate Spearman's rank correlation coefficient between HHI and the search engine revenue over seven weeks for different sectors. The results are summarized in Table 4.1. For each sector, we average the user clicks and advertisers' consumption each week, and then calculate HHI of each sector (sector is a "market" here). It is evident from the table that there is a very strong negative correlation between the HHI for a sector and the revenue the search engine obtained from that sector. This indicates that an increase or decrease in the market competition can lead to a corresponding rise or drop in the search provider's revenue as well. Therefore, it is critical that search engine providers identify scenarios where market competition is becoming stagnant or decreasing so that it can come up with remedial strategies [Boone, 2008], such as free token distributions to intensify the competition.

However, sector level analysis offers a very broad macroscopic view. It is difficult to gain relevant insights into the market competition at the sector level. For example, a low HHI value does not suggest that all keywords within that sector are facing high competition, and vice versa. Thus, it is much more useful for search providers to find different pockets of advertisers and their corresponding keywords *within* a sector and track their competition over time. Our proposed community detection algorithm *Max-Intensity* identifies these competitive advertiser communities and their corresponding bidding keywords.

Table 4.1: Market competition versus search provider revenue

Sector	Criteria	Weekly Averaged Data							Correlation
Instruments	HHI	0.0006	0.0007	0.0007	0.0008	0.0019	0.0017	0.0006	-0.9611
	Revenue	6,059,822	5,876,791	5,494,558	4,614,660	1,593,922	625,657	5,196,103	
Finishing Materials	HHI	0.0009	0.0010	0.0012	0.0025	0.0027	0.0035	0.0011	-0.9417
	Revenue	7,438,719	7,143,498	6,151,826	4,988,395	2,103,629	1,740,237	6,392,471	
Industrial Chemicals	HHI	0.0010	0.0011	0.0011	0.0014	0.0018	0.0026	0.0012	-0.9533
	Revenue	3,175,138	3,042,216	2,862,213	2,347,107	951,526	574,986	2,760,362	
Machinery	HHI	0.0003	0.0004	0.0004	0.0005	0.0008	0.0009	0.0004	-0.9976
	Revenue	5,397,096	5,196,760	4,721,726	3,909,659	1,363,837	789,953	4,650,112	
Metallic Materials	HHI	0.0004	0.0004	0.0005	0.0006	0.0011	0.0016	0.0006	-0.9415
	Revenue	3,441,036	3,330,836	2,920,052	2,177,350	701,702	433,938	3,024,788	
Specialty Hospital	HHI	0.0006	0.0006	0.0006	0.0007	0.0010	0.0014	0.0007	-0.9580
	Revenue	81,584,689	79,453,651	76,563,899	66,196,751	34,401,222	23,445,646	71,740,014	

4.1.2 Data Modeling and Definitions

In this section, we first describe our data model and the intuition behind it. Then we formally define the *intensity score* and evaluate its boundary conditions.

4.1.2.1 Data Modeling

In order to deploy the community detection algorithms, we need to model the sponsored search market as a graph. In our model, we consider both keywords and advertisers as vertices. In this keyword-advertiser graph, an edge exists between a keyword and an advertiser, if the advertiser bids for that keyword and has paid some remuneration to the search provider for it. If an edge exists between an advertiser and a keyword, we say that the advertiser "consumed" the keyword and the corresponding "consumption" is measured by the amount paid by the advertiser to the search provider for that keyword. Since, in our model, there cannot be any advertiser-advertiser or keyword-keyword edges, we essentially have a bi-partite graph. However, all keywords are not equally important to an advertiser. Therefore, we assign weights to the advertiser-keyword edge according to the remuneration the advertiser pays to the search provider for that keyword.

There is a significant drawback if we treat only advertisers as vertices in the graph. This is because the edges can be established between two advertisers vertices, if there is at least one

mutual keyword that both these advertisers have consumed. In this case, there will be too many cliques which may not be meaningful communities. For example, consider a network where five advertisers share a common keyword, two among them share two additional keywords while the remaining three share only one additional keyword. These five advertisers in this model form a clique. However, even if these five advertisers had just one common keyword, they would still have formed a clique. Thus we will end up having too many redundant cliques that goes against the definition of communities (less external connections and more internal connections).

We now formally denote the weighted bi-partite advertiser-keyword graph by $G(A, K, E)$, where A is the set of *advertisers*, K is the set of *keywords* and E is the weighted edge set such that $A \cap K = \varnothing$ and $E \subseteq V \times K$. Let w_{ij} denote the weight of an edge between vertex $i \in A$ and vertex $j \in K$. In the context of our model, $w_{ij} > 0$ is the proportion of the money that advertiser i spent on keyword j. Thus we have

$$w_{ij} = csm_{ij} \times \frac{1}{\sum_{k=1}^{n_i} csm_{ik}} \tag{4.3}$$

where csm_{ij} (shown with red numbers in Figure 4.1) represents the consumption of advertiser i on keyword j and n_i is the total number of keywords consumed by advertiser i.

4.1.2.2 Competition Coefficient

Community detection methods [**?**] partition vertices in a graph into set of groups, also called communities, based on their inter-relationships. Let the subgraph $C(A_C, K_C, E_C)$ of an advertiser-keyword graph $G(A, K, E)$ be such a *community*. In order to measure the degree to which advertisers within a community tend to compete with each other, we propose a *competition coefficient* based on the internal competition within a community.

Definition 1 (Homogeneous Neighborhood)**.** *For a given vertex $u \in A_C \cup K_C$ in a bipartite subgraph C, we define its homogeneous neighbor set as $N(u) = \{v | (u,t) \in E_C \wedge (t,v) \in E_C, t \neq \varnothing, u \neq v\}$, which is a collection of homogeneous vertices.*

Definition 2 (Competition Coefficient)**.** *In a given bipartite subgraph $C(A_C, K_C, E_C)$, the com-*

petition coefficient of an advertiser vertex $i \in A_C$ is defined as follows:

$$cc_i = \begin{cases} \frac{\sum_j \sum_k w_{jk}}{|N(i)|}, & \textit{if } N(i) \neq \varnothing \\ 0 & \textit{if } N(i) = \varnothing \end{cases} \tag{4.4}$$

where $j \in N(i)$, $k \in K_C$, $w_{ik} > 0$ and $w_{jk} > 0$.

Leveraging the concept of clustering coefficient in graph theory [Watts and Strogatz, 1998], we propose the competition coefficient to factor the competition an advertiser vertex will face in a bipartite graph. The definition shows that competition coefficient is the sum of the weighted edges of all the competitors that are bidding on the same keywords as the advertiser, normalized by its number of competitors. Consider the example given in Figure 4.1, the competition coefficient of vertex *HTC* in the community is $cc_{HTC} = \frac{0.7+0.1+0.6+0.4}{3} = 0.6$.

From this definition we can obtain that $cc_i \in [0, 1]$. The maximum value of 1 is obtained when all the homogeneous neighbors spend all their money on the same keywords that are bid on by advertiser i. The lower bound of competition coefficient is 0, which is obtained when no competitors exist ($N(i) = \varnothing$).

Theorem 3. *Given a community $C(A_C, K_C, E_C)$, advertiser $i \in A_C, cc_i = \varphi$. If a new competitor j is introduced in this community, the competition coefficient will increase if and only if*

$$\sum_{w_{ik}>0} w_{jk} > \varphi. \tag{4.5}$$

Proof. Let, $\sum_l \sum_k w_{lk} = \alpha$, where $l \in N(i), k \in K_C$. We have $\varphi = cc_i = \frac{\sum_l \sum_k w_{lk}}{|N(i)|} = \frac{\alpha}{|N(i)|}$. After adding a new competitor j, $\varphi' = \frac{\alpha+\sum_{w_{ik}>0} w_{jk}}{|N(i)|+1}$. If the competition coefficient increases, $\varphi' > \varphi$, then $\frac{\alpha+\sum_{w_{ik}>0} w_{jk}}{|N(i)|+1} > \frac{\alpha}{|N(i)|}$. Thus we can get $\sum_{w_{ik}>0} w_{jk} > \frac{\alpha}{|N(i)|} \times (|N(i)| + 1) - \alpha = \frac{\alpha}{|N(i)|} = \varphi$. □

4.1.3 Intensity Score

Based on the competition coefficient, we formulate the *intensity score* for an advertiser vertex. We assume that if the proportion of advertiser consumption for a keyword is high, then the advertiser's competitive capacity to this keyword is also high. To measure the intensity score of an advertiser within a community, we use the following two criteria:

1. The internal consumption proportion $\sum_{i \in A_C \wedge j \in K_C} w_{ij}$ of the advertiser i inside community C should be more than the maximum consumption proportion to a single external community $\max_{C'} \sum_{i \in A_C \wedge j' \in K_{C'}} w_{ij'}$. This criteria is represented in the intensity computation as the difference between the internal consumption proportion and the maximum external community consumption proportion $\sum_{i \in A_C \wedge j \in K_C} w_{ij} - \max_{C'} \sum_{i \in A_C \wedge j' \in K_{C'}} w_{ij'}$, where C' represents the external communities (more details in case study 1 of Section 4.1.4.3). This value will be between -1 when there are no competitors inside the community, and 1 when there are no competitors outside the community. This criteria emphasizes that a vertex is likely to be within a community if its sum of consumption proportions (edges) within the community is large (see Theorem 1).

2. Within a specific community, if the competition intensity is high, then the homogeneous neighbors of the advertiser vertex i should spend more money on the same keywords as i. In this case, its competition coefficient, cc_i, should be high. (more details in case study 2 of Section 4.1.4.3). This value ranges from 0 (not competitive) to 1 (full competition).

We aggregate these two criteria to formulate *intensity score* I_i of an advertiser vertex i in community $C(A_C, K_C, E_C)$ as follows:

$$I_i = cc_i + \lambda \left(\sum_{i \in A_C \wedge j \in K_C} w_{ij} - \max_{C'} \sum_{i \in A_C \wedge j' \in K_{C'}} w_{ij'} \right) \tag{4.6}$$

where $\lambda \geq 0$ is a tuning parameter. We test different values for λ in our experiments. Base on this formulation, the intensity score for HTC in Figure 4.1 is $I_{HTC} = cc_{HTC} + \lambda(0.3 + 0.5 - 0.2) = 1.2$ (we choose $\lambda = 1$ here).

The intuition behind this intensity measure is based on our observations that an advertiser vertex with high *intensity score* has lots of competitors within its community, and its internal consumption proportion is higher than the maximum consumption proportion to any single external community.

4.1.3.1 Boundary Conditions of Competition Intensity

For vertices that do not have any external connections, I_i is equal to the sum of internal coefficient and λ (i.e. $I_i = cc_i + \lambda$). I_i attains its maximum value of $1 + \lambda$ when advertiser i has no external connections and faces full competition (all competitors spend all their money on the same keywords as advertiser i) inside the community. The theoretical lower bound of I_i is $-\lambda$, which is obtained when the competitors of advertiser i are only from outside of the community. For example, this is possible for singleton advertiser community which has only external connections. Therefore, for every advertiser vertex i, $I_i \in [-\lambda, \lambda + 1]$.

The intensity score of the community $C(A, K, E)$ is given by $I_C = \sum_{i \in A_C} \frac{1}{|A|} I_i$, $I_C \in [-\lambda, \lambda + 1]$. I_C will be closer to $\lambda + 1$ as more vertices inside the community high intensity score. This can happen only when the community has a strong internal structure without any external connections to its vertices and all the keywords inside the community are bid by all the advertisers. If C is a singleton community which has only one advertiser and has only external connections, then $I_C = -\lambda$.

4.1.4 Proposed Approach

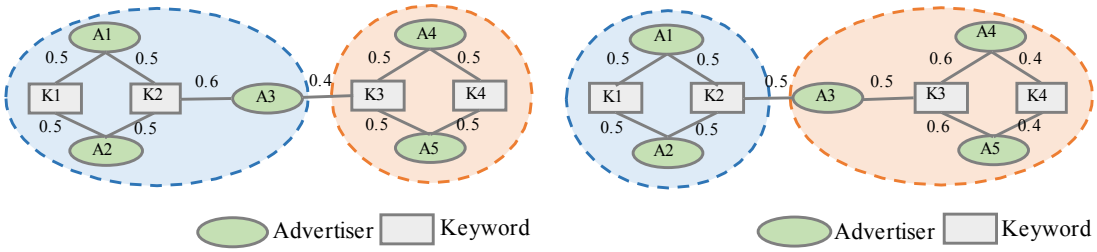
In this section, we present the formal definition of competition community detection problem, and develop a community detection algorithm called *Max-Intensity* that identifies communities by maximizing their intensity scores.

4.1.4.1 Objective

Our objective is to deal with the following problem: given a weighted bipartite graph $G(A, K, E)$, partition the vertices of G into subsets so as to maximize the *intensity score* in each detected community. The goal function is,

$$f_{obj} = \max \sum_{A_C \subseteq G} \sum_{i \in A_C} I_i \tag{4.7}$$

where $C(A_C, K_C, E_C)$ is the detected community. An edge between A and K in $G(A, K, E)$ amounts to an advertiser $a \in A$ bidding on a particular keyword $k \in K$. The weight of an edge



(a) Case study 1: Advertiser assignment regarding to the consumption proportion

(b) Case study 2: Advertiser assignment regarding to the competitors

Figure 4.2: Case studies on advertiser assignment performance of *Max-Intensity*

$W_{a,k}$ represents the consumption of that particular bid normalized by total consumption for the advertiser. With this input set then the algorithm is set to iteratively assign keyword and advertiser vertices to communities in order to increase the overall intensity of the network.

4.1.4.2 Community Detection with Max-Intensity

Similar to finding subgraphs of a given size with some fitness measures larger than a threshold [Fortunato, 2010], our problem is also NP-complete. Therefore, we utilize a heuristic approach which strives to obtain a high value of intensity. To achieve this desiderata, the algorithm iterates through the entire advertiser set and tests the assignment of an advertiser to each of its neighboring communities against no assignment at all. The advertiser is then assigned to the community which results in the highest intensity for the graph overall (pseudocode in Algorithm 2). After each assignment outside of an advertiser's current community, and initially, each keyword is assigned to the community which has the highest average intensity score, as shown in Algorithm 3. An iteration limit is set so as to avoid infinite looping in the event that the assignment of vertices oscillate. By iteratively assigning the advertiser and keyword vertices the algorithm will approach a network with a good intensity score.

Algorithm 2 Max Intensity

Input: A weighted bipartite graph G.

Output: Intensity of advertisers in G; Detected communities.

procedure MAXINTENSITY($G(A, K, E), \lambda$)
  ASSIGNKEYWORDS(K) ▷ Each vertex is assigned to its seed community
  $Sum \leftarrow -\lambda \times |A|$, $Old\_Sum \leftarrow -1$
  do
    $Old\_Sum \leftarrow Sum$, $Sum \leftarrow 0$
    for $a \in A$ **do**
      $cur\_i \leftarrow Intensity(a)$
      if $cur\_i == \lambda + 1$ **then**
        $Sum \leftarrow Sum + cur\_i$
        continue
      end if
      $cur\_i\_neig \leftarrow 0$
      for $l \in Neig(Neig(a))$ **do**
        $cur\_i\_neig \leftarrow cur\_i\_neig + Intensity(l)$
      end for
      for $C \in Comm(Neig(a))$ **do**
        $n\_i \leftarrow Intensity(a)$ ▷ Move a to community C
        $n\_i\_neig \leftarrow 0$
        for $l \in Neig(Neig(a))$ **do**
          $n\_i\_neig \leftarrow n\_i\_neig + Intensity(l)$
        end for
        if ($cur\_i < n\_i$) and ($cur\_i\_neig < n\_i\_neig$) **then**
          $cur\_i \leftarrow n\_i$, ASSIGNKEYWORDS(K)
        else
          replace a to its original community
        end if
      end for
      $Sum \leftarrow Sum + cur\_i$
    end for
  while not stopping criterion and $Sum \neq Old\_Sum$
  return $Advertiser\_intensity = Sum/|A|$
end procedure

Algorithm 3 Assign Keywords

Input: A partite set of keywords K.

Output: Assigned communities of K

procedure ASSIGNKEYWORDS(K)
  for all $k \in K$ **do**
    $max\_i \leftarrow -\infty$
    for $C \in Comm(Neig(k))$ **do**
      $cur\_i \leftarrow 0$, $A \leftarrow \{\, a \mid (a \in Neig(k)) \wedge (a$ is a memeber of $C) \,\}$
      for $a \in A$ **do**
        $cur\_i \leftarrow cur\_i + Intensity(a)$
      end for
      $cur\_i \leftarrow \frac{cur\_i}{|V|}$
      if $cur\_i > max\_i$ **then**
        $max\_i \leftarrow cur\_i$, and move k to community C
      end if
    end for
  end for
end procedure

4.1.4.3 Case Study

In this part, we study the behavior of *Max-Intensity* algorithm with two simple cases. In the first case study, we test the advertiser assignment regarding the consumption proportion. The initial choice of vertices in the graph is arbitrary but suppose without loss of generality the vertices are chosen in order (i.e. $A1, A2, ..., A5$). Initially each vertex is assigned to it's seed community ($C1, C2, ..., C5$). The intensity score of an advertiser vertex in it's seed community is $-\lambda$ due to hitting the lower boundary condition (Section 4.1.3.1). Then the keywords are assigned to the community with the highest average intensity score. Again without loss of generality assume ties are assigned to the lower ordered vertex (i.e $A1 < A2 < ..., A5$). There are other intutive choices to make in breaking ties, such as assigning the keyword to the advertiser with a higher weight, but for simplicity we will just use this basic ordering. After initialization, each advertiser is assigned to a distinct community with $K1$ and $K2$ assigned to $A1$'s community ($C1$), $K3$ assigned to $A3$'s community ($C3$) and $K4$ assigned to $A4$'s community ($C4$). $C2$ and $C5$ are still singleton com-

munities with $A2$ and $A5$ respectively. The algorithm begins the iteration, choosing $A1$ first. The intensity score will be calculated at this vertex and is given to be 1 because all attached keywords are within the same community. Next $A1$ will stay in $C1$ since all other assignments will lead to a minimum score. This is because all attached keywords are in $C1$, so an assignment to another community will lead to $cc_{A1} = 0$ ($N(A1) = \varnothing$). The intensity score thus becomes $-\lambda$, the left-most bound. The next vertex chosen will be $A2$. The initial score for $A2$ is -1 because that is the maximum weight of the leaving edges to $C1$. Once $A2$ is placed into $C1$ then the intensity will increase and $A2$ will stay in $C1$. This will result in a call to $Assign\_Keywords(K)$ will place $K3$ into $C4$. $A3$ is next and will follow the same steps as $A2$, this time choosing $C4$. $A4$ will stay in $C4$ by the same reasoning as $A1$. The final advertiser, $A5$ is placed into $C4$ because it will increase the intensity score of the graph, ending the current iteration. The next iteration begins selecting the advertisers in order. $A1$ and $A2$ will remain in $C1$ since all of their keywords are contained in $C1$. $A3$, which at this time is in $C4$, has the choice to stay or move to $C1$. Since the intensity is higher for both $A3$ and $A3$'s neighbors to switch, $A3$ will be placed in $C1$. The remaining two advertisers will stay in their current communities. Since there are no other options, the algorithm will terminate and the result is shown in Figure 4.2(a).

In the second case study, we test the advertiser assignment regarding the competitors in the same community. The iteration will proceed as in the first case for the first iteration. On the second iteration, as in the first case, $A1$ and $A2$ will stay in their communities. Once the vertex $A3$ is taken, the vertex will stay in it's own community rather than adding it to $C1$ which will not increase the intensity of $A3$. The algorithm will proceed and will not change the communities of $A4$ and $A5$. This leaves the network partitioned into two communities as shown in Figure 4.2(b).

4.1.4.4 Effectiveness Analysis

To help in analyzing the complexity of the algorithm, the shorthand notation that is presented along with a description in Table 4.2 will be used.

Since the algorithm runs in a max total of M times in the worst case this factor is run for each vertex assignment iteration. In the vertex assignment iteration the most dominate time fac-

Table 4.2: Notations used in effectiveness analysis

Notation	Description
$\|V\|$	number of vertices in G
$\|A\|$	number of advertiser vertices in V
$\|K\|$	number of keyword vertices in V
$\|E\|$	number of edges in G
$\|C\|$	number of communities in G
M	max iteration

tor comes from iterating through all the communities in the keyword edge set neighbors of an advertiser. This additional factor of $|C|$ will be multiplied to the intensity calculation of the neighbors of the adjoining advertisers of the keyword. This cost will be dominated by a move to a new community which will call the $Assign\_Keywords$ subroutine. Since the $Assign\_Keywords$ subroutine iterates over all keywords this factor of $|K|$ will be multiplied by the run through all communities multiplied by the intensity score of each advertiser neighbor. This total cost so far is $O(M \times |K| \times |C| \times |A| \times |C| \times cost(i))$, where $cost(i)$ is the cost of intensity score calculation which is dominated by the cost associated with calculating the competition coefficient and in the worst case is $|K| \times |A|$ giving a total worst case running time of $O(M \times |K| \times |C| \times |A| \times |C| \times |K| \times |A|) \Rightarrow O(M \times |K|^2 \times |C|^2 \times |A|^2)$. Although this running time is slow in the worst case, actual running times are much faster. This is due to the time complexity reaching its worst case only when an assignment to a new community is made which has the effect of reducing $|K|$, $|C|$, $|A|$ in subsequent iterations.

4.1.5 Evaluation

In this section, we provide a brief overview of our sponsored search advertising dataset collected from one search provider, the comparative methods and the evaluation metrics that we used in our experiments, and finally the performance analysis of our *Max-Intensity* algorithm.

Table 4.3: Daily sponsored search advertising dataset statistics

Network	Dataset	Average Statistics
$\|V\|$	advertisers	$\approx 200,000$
	keywords	$\approx 2,000,000$
$\|E\|$	consumption records	$\approx 6,000,000$

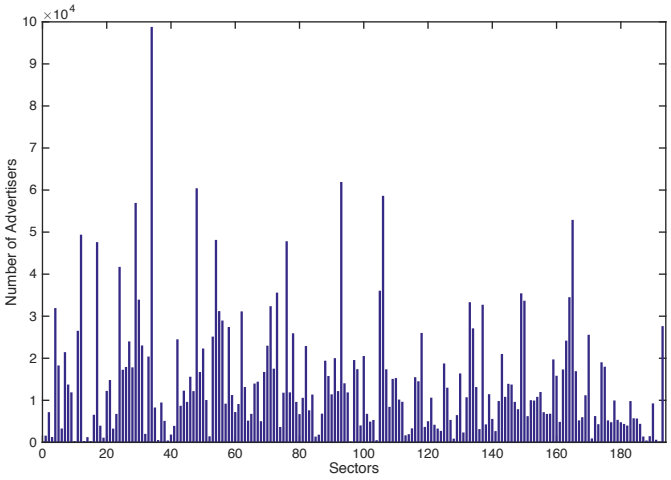
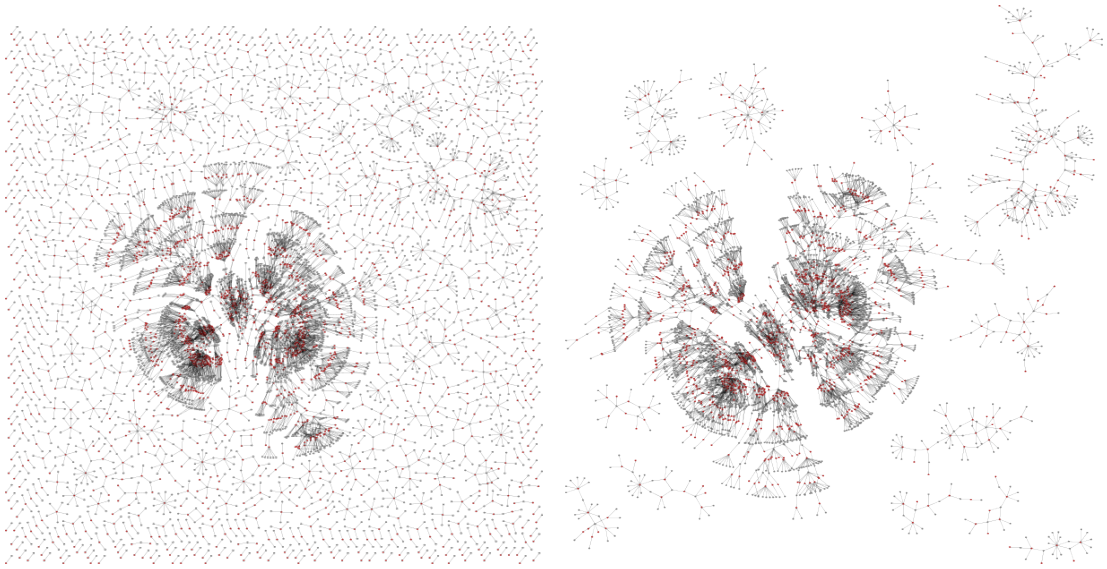


Figure 4.3: Advertiser distribution among all sectors

4.1.5.1 Dataset Description

We conducted all our experiments on actual sponsored search advertising data collected for a period of two months. These datasets are structured and capture information about different advertisers in different sectors consuming various keywords. The overall data is available at two different granularities–one at the keyword level and the other at the advertiser level. This data has been curated and anonymized to ensure business secrets and privacy information are not publicly available. For example, we do not know the exact keywords, but instead we have the keyword ids. This, however, does not concern us, since our model and community detection methodology do not require this information. Table 4.3 shows the average statistics for daily sponsored search advertising dataset.

There are about 200 sectors in this dataset like IT, electronics, food, nourishment, etc. Figure 4.3 shows the advertiser distribution among all sectors. In our experiment, we selected the top



(a) Input network before filtering (b) Input network after filtering

Figure 4.4: Input network pre-processing

four popular sectors.

Pre-processing. Given the nature of the dataset, modeling of the data can result in a set of highly disconnected and sparse subgraphs which act as noise in the mining analysis. This is due to sets of keywords being exclusively bid on by a small set of advertisers. Therefore, prior to constructing the advertiser-keyword graph, we pre-process the input to reduce the noise. We filter out the noise by building a set of trees using breadth-first search (BFS) [Moore, 1959] over the entire vertex set of the graph and then considering only those trees whose depth and vertex set size are above user-defined threshold. We demonstrate the filtering process through an example shown in Figure 4.4(a) and Figure 4.4(b). It is apparent from the figures that the strongly connected clusters in our advertiser-keyword graph are retained, while those subgraphs, which act as outliers, are removed.

4.1.5.2 Comparative Methods and Evaluation Metrics

In this subsection, we consider the canonical community detection algorithms as well as recent state-of-the-art algorithms. We used the *python-igraph* package for executing the canonical al-

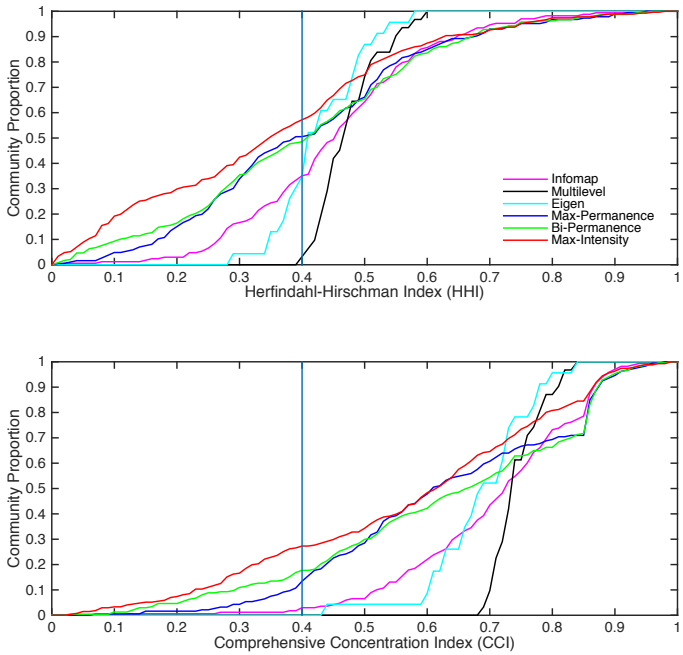


Figure 4.5: Cumulative proportion of competition communities detected by each method

gorithms *Infomap*, *Multilevel* and *Leading Eigenvector*, while we implemented *Max-Permanence*, *Bi-Permanence* and *Max-Intensity* algorithms in C#. We then evaluate the quality of the detected communities by calculating their Herfindahl-Hirschman index (HHI) [Hirschman, 1964] and comprehensive concentration index (CCI) [Horvath, 1970]. According to the definition of HHI and CCI mentioned in Section 4.1.1, smaller index scores indicate more competitive communities.

4.1.5.3 Experimental Results

In this section, we first evaluate the competitiveness of the communities detected from different algorithms by comparing their HHI and CCI scores. Then we test the performance of our Max-Intensity algorithm for different λ values. Lastly, we examine the computational complexity of our Max-Intensity algorithm.

Community Competitiveness Evaluation. To compare the competitiveness of communities detected by different algorithms, we run our experiments on one month dataset for the four sectors mentioned previously. Here we choose $\lambda = 1$ for all our experiments.

Figure 4.5 shows the cumulative proportion of communities against different HHI and CCI

Table 4.4: Community number and average HHI/CCI of each method

Methods	#Comminities	Average HHI	Average CCI
Infomap	164	0.4542	0.7100
Multilevel	31	0.4730	0.7439
Eigen	23	0.4278	0.6868
Max-Permanence	271	0.4050	0.6292
Bi-Permanence	237	0.4008	0.6264
Max-Intensity	186	**0.3488**	**0.5725**
Average	152	0.4183	0.6615

values for various algorithms averaged across four sectors. In this figure, x-axis represents the HHI (above) and CCI (below) values while y-axis represents the cumulative community proportion accordingly (averaged over four sectors). For example, a point $(0.4, 0.6)$ on the curve means 60% of all the detected communities by a certain method has HHI or CCI ≤ 0.4. According to the definitions, smaller HHI and CCI values indicate more competitive communities. We can see from the figure that the cumulative proportion of competitive communities detected by our algorithm *Max-Intensity* is significantly higher than other methods, particularly for lower HHI or CCI values (left of the vertical line, HHI=0.4 or CCI=0.4). The nature of the curves for *Max-Intensity*, *Bi-Permanence*, *Max-Permanence* and *Infomap* are very similar. However, the curves corresponding to *Multilevel* and *Eigen* rise steeply from moderate HHI values and ends abruptly. This is because, both these methods detect very few communities, each with a large number of community members, which fail to capture the actual competition in the marketplace. In other words, both *Multilevel* and *Eigen* cannot detect either intense competitive communities or communities without significant competition, instead they merge smaller community structures to get an overall moderate HHI. This is primarily because modularity-based community detection algorithms (like *Multilevel* or *Eigen*) suffer from resolution limit problems and end up selecting very few but large communities. This is conclusively shown in table 4.4.

Table 4.4 displays the average number of communities detected by each method and their

Table 4.5: Cumulative proportion of competition communities under different HHI/CCI threshold

Methods	HHI			
	≤ 0.1	≤ 0.2	≤ 0.3	≤ 0.4
Infomap	1.62%±0.0060	4.55%±0.0147	17.02%±0.0242	39.43%±0.0688
Multilevel	0.00%±0.0000	0.00%±0.0000	0.00%±0.0000	10.88%±0.0655
Eigen	0.00%±0.0000	1.76%±0.0214	2.85%±0.0205	25.83%±0.0640
Max-Permanence	4.36%±0.0128	15.58%±0.0269	32.60%±0.0339	48.25%±0.0322
Bi-Permanence	10.45%±0.0175	19.93%±0.0355	34.80%±0.0340	53.21%±0.0417
Max-Intensity	**16.02%**±0.0218	**27.44%**±0.0188	**41.88%**±0.0111	**59.52%**±0.0170
Methods	CCI			
	≤ 0.1	≤ 0.2	≤ 0.3	≤ 0.4
Infomap	0.15%±0.0031	1.00%±0.0047	1.47%±0.0031	4.09%±0.0071
Multilevel	0.00%±0.0000	0.00%±0.0000	0.00%±0.0000	0.00%±0.0000
Eigen	0.00%±0.0000	0.00%±0.0000	0.68%±0.0135	1.76%±0.0214
Max-Permanence	0.40%±0.0050	1.35%±0.0062	5.18%±0.0192	13.79%±0.0214
Bi-Permanence	1.73%±0.0051	5.64%±0.0143	12.16%±0.0140	19.12%±0.0392
Max-Intensity	**3.07%**±0.0037	**8.41%**±0.0071	**15.07%**±0.0116	**24.00%**±0.0250

corresponding average HHI and average CCI values (across all the communities). As shown in the table, *Max-Intensity* achieves the lowest average HHI and average CCI values.

We also tabulate the cumulative proportion of competitive communities under different HHI and CCI thresholds with their respective standard deviations from 0.1 to 0.4 with an interval of 0.1 in Table 4.5. We can observe from the table that, when HHI ≤ 0.1, the cumulative proportion of communities detected by *Max-Intensity* is 1.5 to 10 times higher than its closest competitors namely *Infomap*, *Max-Permanence* and *Bi-Permanence*. Similarly, when CCI ≤ 0.1, the cumulative proportion of communities detected by *Max-Intensity* is 1.8 to 20.5 times higher.

Parameter Analysis. The *intensity score* defined by Eq. (4.6) is dependent on the parameter λ. The value of λ decides the relative importance between the competition coefficient and the

difference between advertiser's consumption within the community and outside. If the value of the latter difference is high, it implies that the advertiser has less capacity to compete outside this community. By varying the parameter λ, we want to examine which component of the intensity score is more important to identify competitive communities in the sponsored search market. The special case of $\lambda = 1$ indicates that both these components are equally important. We choose different values of λ from 0.001 to 20 and compute the corresponding average HHI and CCI values. The results are summarized in Figure 4.6. It is evident from the figure that with $\lambda < 1$, we get more competitive communities (with low HHI and CCI values). However, the curves corresponding to $\lambda < 1$ follow a broad "U" pattern. This means that as λ increases from 0 to 1, the average HHI or CCI decreases till it attains its optimal value, after which it starts increasing. In this case, we found that $\lambda = 0.1$ gave the best HHI and CCI values. Likewise, for $\lambda \geq 1$, the average HHI or CCI values sharply increase. Thus, we can infer that the competition coefficient is significantly more important in the definition of the intensity score.

Computational Analysis. Lastly, we examine the computational time for *Max-Intensity* and compare it with the *Bi-Permanence* algorithm which has the second best performance in competitive community detection results, as shown in Figure 4.5. In this experiment, we ran the algorithms on different graphs with varying number of edges. The running times of the algorithms averaged over 10 iterations are plotted against the number of edges in Figure 4.7. The results show that the running time of *Bi-Permanence* is only marginally faster than our *Max-Intensity* algorithm, since these two algorithms have similar time complexity. However, our method *Max-Intensity* can detect more competitive communities than *Bi-Permanence*.

4.2 Summary

This chapter describes a novel approach *Max-Intensity* to detect competitive communities for weighted bi-partite network formed by advertisers and their bidding keywords. This approach iteratively assigns keyword and advertiser vertices to communities in order to increase the overall intensity score of the network. The proposed intensity score takes into consideration two factors: the competitors that bid the same keywords and the advertisers' consumption proportion within

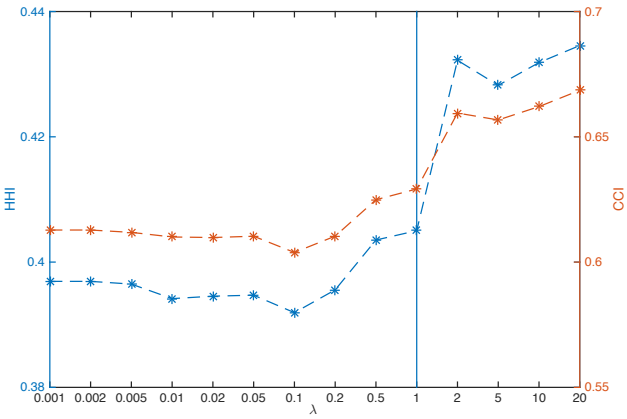


Figure 4.6: Average HHI and CCI for different λ

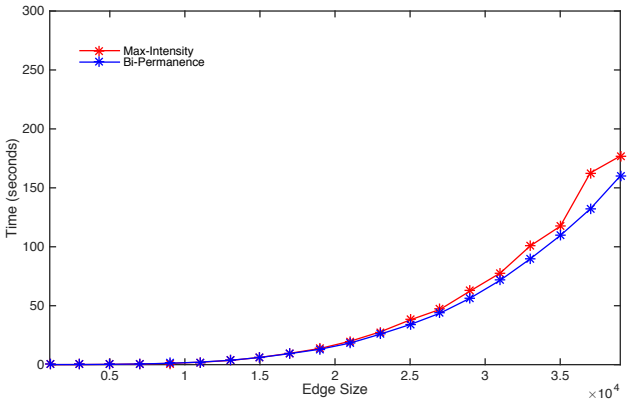


Figure 4.7: Running time comparison

the community. The proposed methods *Max-Intensity* is compared with canonical community detection methods *Infomap* [Rosvall and Bergstrom, 2007], *Multilevel* [Blondel et al., 2008] and *Eigen* [Newman, 2006a], and the state-of-the-art method *Max-Permanence* [Chakraborty et al., 2014] as well as its bipartite version (*Bi-Permanence*). HHI and CCI are employed as evaluation metrics to measure the competition within the detected communities. Compared to these baseline methods, the communities detected by *Max-Intensity* algorithm have the lowest HHI and CCI values, thereby demonstrating that the *Max-Intensity* method identifies more competitive communities.

Part II

Evolutionary Network Analysis

CHAPTER 5

Evolutionary Network Analysis: Methods

The problem of evolutionary network analysis has gained increasing attention in recent years, because of an increasing number of networks, which are encountered in temporal settings, and it is desirable to learn interesting trends about how the network structure evolves over time in terms of other interesting trends. One challenging aspect of networks is that they are inherently resistant to parametric modeling, which allows us to truly express the edges in the network as functions of time. This is because, unlike multidimensional data, the edges in the network reflect interactions among nodes, and it is difficult to independently model the edge as a function of time, without taking into account its correlations and interactions with neighboring edges. In Section 5.1 we show that it is indeed possible to achieve this goal with the use of a matrix factorization, in which the entries are parameterized by time. This approach allows us to represent the edge structure of the network purely as a function of time, and predict the evolution of the network over time. This opens the possibility of using the approach for a wide variety of temporal network analysis problems, such as predicting future trends in structures, predicting links, and node-centric anomaly detection. This flexibility is because of the general way in which the approach allows us to express the structure of the network as a function of time. Inspired by the time-dependent matrix factorization technique, Section 5.2 introduces a framework to model co-evolution across multiple networks with *shared temporal matrix factorization*. It decomposes the adjacency matrix of each co-evolving network into a product of network-independent shared factor and a set of network-specific temporal factors, and impose a non-negativity constraint on the factors for greater interpretability. The benefits of this approach is demonstrated in predicting co-evolutions across multiple networks such as cross-network link prediction, lag correlation detection and community detection.

5.1 Temporally Factorized Network Modeling

Temporal networks have become ubiquitous because of the numerous applications that generate network structures in a time-dependent way. Recently, a significant amount of work has been done in the area of *evolutionary network analysis* [Aggarwal and Subbian, 2014, Ranshous et al., 2015], which examines various problems in the context of network evolution. Some examples of such problems are as follows:

- Based on the trends in the past, which links are most likely to be received at a future point in time? How does the likelihood change with increasing value of time. Note that this is a more refined problem than traditional link prediction [Dunlavy et al., 2011, Sarkar et al., 2012], in which one simply predicts the links based on a static state of the network.

- How do communities evolve over time? Which communities grow, and which ones shrink? Which ones are expected to grow in the future? Numerous works have been proposed in this context [Backstrom et al., 2006, Gupta et al., 2011, Chi et al., 2007], although none of these methods fully capture the evolving nature of the underlying network.

- One would like to predict surprising or anomalous events in different regions of the networks [Akoglu and Faloutsos, 2010, Ide and Kashima, 2004, Yu et al., 2013a]. These could represent sudden regions of change [Yu et al., 2013a], or other structural changes in the network [Gupta et al., 2011].

Although many individual solutions exist for these problems, a broader question is whether we can directly characterize the structure of the network as a function of time. The ability to characterize the structure of the network as a function of time is crucial in using it in different application settings, because such a characterization can capture very rich information about the structure of the underlying network. In this section, we discuss one such model, with the use of matrix factorization methods.

Matrix factorization is a natural method to express the evolutionary structure of networks because of its ability to leverage the structural correlations among the edges in the network. The basic

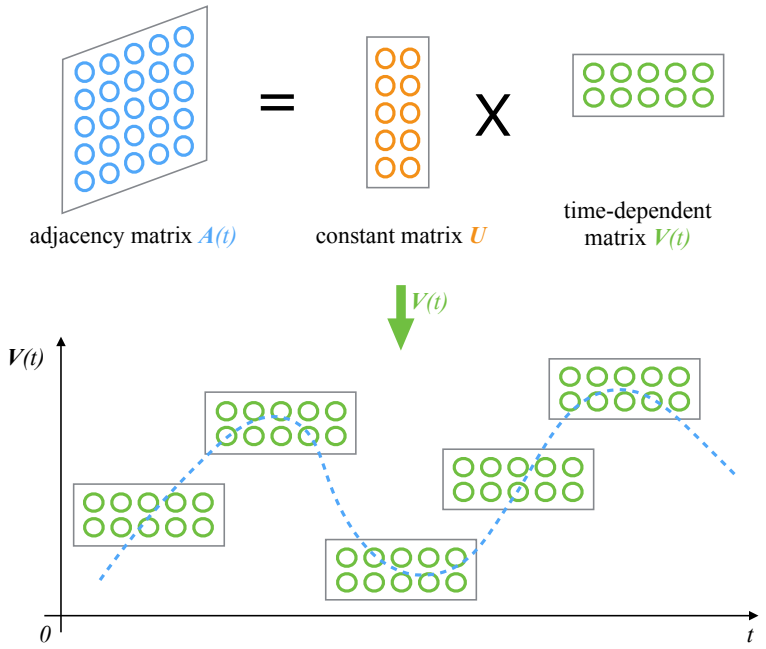


Figure 5.1: Illustration of temporal matrix factorization model

idea of temporal matrix factorization methods is to extract a low rank representation of the underlying adjacency matrices, in a way which are parameterized with time, as shown in Figure 5.1. This temporally parameterized factorization can be used to reconstruct the structure of the network at any time t, including at times in the past or future where the network has not been observed. This ability to reconstruct the adjacency structure of the network at any time t is crucial; it allows one to make far more general predictions. Furthermore, it can be viewed as a compressed representation of not just the current state of the network, but the *entire dynamic profile of the network over time*. This comprehensive characterization is crucial in enabling an effective solution to a variety of problems. We view our solution as *generic*, as it is not specific to a particular problem, but it enables solutions across a wider variety of settings.

5.1.1 The Temporal Matrix Factorization Model

In the following, we will generally assume that we have a temporal network $G(t) = (N, \boldsymbol{A}(t))$, where N is set of nodes in the networks, and $\boldsymbol{A}(t)$ is the adjacency matrix of the edges, as a function of time. We assume that the size of $\boldsymbol{A}(t) = [a_{ijt}]$ is $n \times n$, where $|N| = n$. For

unweighted networks, the matrix $\boldsymbol{A}(t)$ is binary, whereas for weighted networks, the matrix $\boldsymbol{A}(t)$ might contain arbitrary weights which change with time. For example, in the case of the *DBLP* network, a_{ijt} might represent the number of publications between authors i and j at time t. These weights might even be negative for dynamic signed networks such as a dynamic *Epinions* network with shifting trust relationships. In general, the way in which $\boldsymbol{A}(t)$ changes will depend on the specific application at hand, and it is completely agnostic to the model discussed in our paper. It is assumed that the time stamp t is a continuous variable that varies from 1 through T.

Note that the set of nodes in the current state of the network is also a function of time, but in practice, one can only work with the set of nodes that one has seen so far historically. Therefore, the set N is fixed to the *union* of all nodes received so far at the current time t, at which the analysis is performed. It is generally relatively easy to also provide estimations of the number of new nodes that various parts of the network can receive as neighbors in the future. If the network $G(t)$ is directed, the basic temporal rank-k matrix factorization model assumes that the matrix $\boldsymbol{A}(t)$ can be factorized as follows:

$$\boldsymbol{A}(t) = f(\boldsymbol{U}\boldsymbol{V}(t)^T) \tag{5.1}$$

Here, both $\boldsymbol{U}$ and $\boldsymbol{V}(t)$ are $n \times k$ matrices. The main difference between $\boldsymbol{U}$ and $\boldsymbol{V}(t)$ is that $\boldsymbol{U}$ is a constant matrix and $\boldsymbol{V}(t)$ is time-dependent. The function $f(\cdot)$ is an element-wise function on elements of the matrix in $\boldsymbol{U}\boldsymbol{V}(t)$, which is useful in certain settings. For example, if the elements in $\boldsymbol{A}(t)$ are normalized to the range $(0, 1)$, then one can use a logistic function for $f(\cdot)$.

Obviously, results from Eq. (5.1) are trivially generalizable to undirected networks, since undirected networks are special cases of directed networks. For undirected networks, the matrix $\boldsymbol{A}(t)$ is symmetric. We can simply average $\boldsymbol{U}\boldsymbol{V}(t)^T$ and its transpose as the prediction of $\boldsymbol{A}(t)$. It can also be factorized as the product of a time-dependent matrix $\boldsymbol{V}(t)$ and its transpose:

$$\boldsymbol{A}(t) = f(\boldsymbol{V}(t)\boldsymbol{V}(t)^T) \tag{5.2}$$

Note that one can also make both $\boldsymbol{U}$ and $\boldsymbol{V}(t)$ time-dependent, although the simpler model can often achieve good approximations and avoid overfitting. Furthermore, it is also possible to make $\boldsymbol{U}$ time-dependent instead of $\boldsymbol{V}(t)$ to achieve similar results. It is possible to simplify the aforementioned relation, by not using a functional transformation $f(\cdot)$, and simply expressing $\boldsymbol{A}(t)$

as follows:

$$\boldsymbol{A}(t) = \boldsymbol{U}\boldsymbol{V}(t)^T \quad \text{or} \quad \boldsymbol{A}(t) = \boldsymbol{V}(t)\boldsymbol{V}(t)^T \tag{5.3}$$

The function $\boldsymbol{V}(t)$ can take on any canonical form, such as linear models, polynomial models, and so on. The choice of models is, however, orthogonal to the key ideas in this study as shown in Section 5.1.1.1.

How does one determine the values of $\boldsymbol{U}$ and $\boldsymbol{V}(t)$? The standard approach in matrix factorization is to set up a least squares optimization problem, so that the $\boldsymbol{A}(t)$ matches $f(\boldsymbol{U}\boldsymbol{V}(t)^T)$ as closely as possible. This can be achieved by minimizing the sum of the squares of the entries in $\boldsymbol{A}(t) - f(\boldsymbol{U}\boldsymbol{V}(t)^T)$. Therefore, one can express this optimization problem as the minimization of the time-decayed sum of the Frobenius norms of the matrix $\boldsymbol{A}(t) - f(\boldsymbol{U}\boldsymbol{V}(t)^T)$ over all values of t from 1 to the current time T.

$$\min_{\boldsymbol{U},\boldsymbol{V}} J(\boldsymbol{U}, \boldsymbol{V}) = \sum_{t=1}^{T} \frac{D(t)}{2} \|\boldsymbol{A}(t) - f(\boldsymbol{U}\boldsymbol{V}(t)^T)\|_F^2 \tag{5.4}$$

Here, $D(t)$ is a decay function with time t that regulates the greater importance of the current state of the network with respect to the past time stamps. For example, one might choose the decay function as the exponential decay function with parameter $\theta > 0$:

$$D(t) = e^{-\theta(T-t)} \tag{5.5}$$

One challenge with the use of this approach is that the network may be very large, and only a small number of edges may be present in $\boldsymbol{A}(t)$. For a network with n nodes, the $O(n^2)$ objective function expressed above might simply be computationally too expensive to even represent effectively. In such cases, the presence of an edge between a pair of nodes in $\boldsymbol{A}(t)$ is more significant than the absence of an edge. The absence of an edge, in fact, often conveys far more noisy information in real settings. Therefore, the aforementioned objective function should be tailored to edge presence rather than edge absence. However, we do need a sample of absent edges to properly train the model. Let $S(t)$ be a sample of edges (i, j) at time-stamp t such that the value of a_{ijt} is 0. At time t, let $E(t)$ be the set of edges for which the weights in $\boldsymbol{A}(t)$ are non-zero at time t. Therefore, we have the following:

$$E(t) = \{(i, j)|a_{ijt} > 0\} \cup S(t) \tag{5.6}$$

Note that the size of the set $E(t)$ is much smaller than $O(n^2)$ because real networks are sparse in real settings. Then, one can express the aforementioned objective function in terms of the edges that are present in the network as follows:

$$J(\boldsymbol{U}, \boldsymbol{V}) = \sum_{t=1}^{T} \frac{D(t)}{2} \sum_{(i,j)\in E(t)} (a_{ijt} - f(\boldsymbol{U}\boldsymbol{V}(t)^T)_{ij})^2 \tag{5.7}$$

Similarly, for undirected network we have

$$J(\boldsymbol{V}) = \sum_{t=1}^{T} \frac{D(t)}{2} \sum_{(i,j)\in E(t)} (a_{ijt} - f(\boldsymbol{V}(t)\boldsymbol{V}(t)^T)_{ij})^2 \tag{5.8}$$

Note that the number of terms in this objective is dependent on the number of edges in the network, which is much easier to handle in practical settings. We also need to add a regularization term to reduce the model variance. However, the specific regularization term will be discussed in the next section, because it depends on the choice of the temporal function $\boldsymbol{V}(t)$.

5.1.1.1 Model Choices

In this section, we will set up various forms of the model for various choices of $f(\cdot)$, and $\boldsymbol{V}(t)$. One typical choice for $f(\cdot)$ include the use of the identity function, and that for $\boldsymbol{V}(t)$ is a polynomial function. In such a case, we can represent $\boldsymbol{V}(t)$ as follows:

$$\boldsymbol{V}(t) = \boldsymbol{W}^{(0)} + \boldsymbol{W}^{(1)}t + ... + \boldsymbol{W}^{(d)}t^d = \sum_{i=0}^{d} \boldsymbol{W}^{(i)}t^i \tag{5.9}$$

Here $\{\boldsymbol{W}^{(i)}\}_{i=0}^{d}$ are all $n \times k$ matrices, which need to be learned from the model along with $\boldsymbol{U}$. $d \in N_+$ and $d \geq 1$, when $d = 1$, $\boldsymbol{V}(t)$ is the simplest linear function. Given the definition of $\boldsymbol{V}(t)$, the matrix form of Eq. (5.7) becomes the following with added regularization:

$$J(\boldsymbol{U}, \boldsymbol{W}) = \sum_{t=1}^{T} \frac{D(t)}{2} \|\mathbf{1}_{E(t)}(\boldsymbol{A}(t) - \boldsymbol{U}(\sum_{i=0}^{d} \boldsymbol{W}^{(i)}t^i)^T)\|_F^2 + \frac{\alpha}{2}\|\boldsymbol{U}\|_F^2 + \sum_{i=0}^{d} \frac{\beta_i}{2}\|\boldsymbol{W}^{(i)}\|_F^2 \tag{5.10}$$

where,

$$\mathbf{1}_{E(t)}(X) = \begin{cases} X_{ij} & \text{if } (i,j) \in E(t), \\ 0 & \text{if } (i,j) \notin E(t). \end{cases}$$

The regularization terms add the Frobenius norms of the matrices $\boldsymbol{U}$ and $\boldsymbol{W}^{(i)}$, so as to minimize overfitting. Similarly, for undirected graphs, we obtain the following loss function:

$$J(\boldsymbol{W}) = \sum_{t=1}^{T} \frac{D(t)}{2} \|\mathbf{1}_{E(t)}(\boldsymbol{A}(t) - (\sum_{i=0}^{d} \boldsymbol{W}^{(i)} t^i)(\sum_{i=0}^{d} \boldsymbol{W}^{(i)} t^i)^T)\|_F^2 + \sum_{i=0}^{d} \frac{\beta_i}{2} \|\boldsymbol{W}^{(i)}\|_F^2 \tag{5.11}$$

5.1.1.2 Model Solutions

In this section, we first compute the partial derivatives of objective functions Eq. (5.10) and Eq. (5.11), with respect to $\boldsymbol{U}$ and $\boldsymbol{W}$ to derive updates. Then we present the temporal matrix factorization algorithms for both asymmetric and symmetric adjacency matrices.

Consider the model for directed networks, one needs to minimize the loss function $J(\boldsymbol{U}, \boldsymbol{W})$,

$$\min_{\boldsymbol{U},\boldsymbol{W}} J(\boldsymbol{U}, \boldsymbol{W}) = \sum_{t=1}^{T} \frac{D(t)}{2} \|\mathbf{1}_{E(t)}(\boldsymbol{A}(t) - \boldsymbol{U}(\sum_{i=0}^{d} \boldsymbol{W}^{(i)} t^i)^T)\|_F^2 + \frac{\alpha}{2} \|\boldsymbol{U}\|_F^2 + \sum_{i=0}^{d} \frac{\beta_i}{2} \|\boldsymbol{W}^{(i)}\|_F^2 \tag{5.12}$$

We introduce a "decayed error term" $\xi(t)$ for each time stamp t, as follows:

$$\xi(t) = D(t)\mathbf{1}_{E(t)}(\boldsymbol{A}(t) - \boldsymbol{U}(\sum_{j=0}^{d} \boldsymbol{W}^{(j)} t^j)^T) \tag{5.13}$$

To compute the gradient, we will need to differentiate our error function. Since our function is defined by parameter matrices $\boldsymbol{U}$ and $\boldsymbol{W}^{(i)}$, we will need to compute a partial derivative for each. These derivatives work out to be,

$$\frac{\partial J(\boldsymbol{U}, \boldsymbol{W})}{\partial \boldsymbol{U}} = \sum_{t=1}^{T} \xi(t)(-\sum_{i=0}^{d} \boldsymbol{W}^{(i)} t^i) + \alpha \boldsymbol{U} \tag{5.14}$$

$$\frac{\partial J(\boldsymbol{U}, \boldsymbol{W})}{\partial \boldsymbol{W}^{(i)}} = \sum_{t=1}^{T} \xi(t)^T(-\boldsymbol{U} t^i) + \beta_i \boldsymbol{W}^{(i)} \tag{5.15}$$

We now have all the derivatives needed to run gradient descent. Pseudocode of the full approach is given in Algorithm 4, where λ is the learning rate. To train this model, we can now repeatedly take steps of gradient descent to reduce our cost function $J(\boldsymbol{U}, \boldsymbol{W})$. Note that TMF is a general framework, which can be adapted for both directed and undirected networks.

Algorithm 4 Algorithm for the TMF model

Input: temporal adjacency matrices $\{\boldsymbol{A}(t)\}_{t=1}^{T}$, the order d of $\boldsymbol{V}(t)$ and latent dimension k.

Output: results of factor matrices $\boldsymbol{U}$ and $\{\boldsymbol{W}^{(i)}\}_{i=1}^{d}$ and the predict adjacency matrix $A(T+1)$.

1: Set k and d.
2: Randomly initialize $\boldsymbol{U}$ and $\{\boldsymbol{W}^{(i)}\}_{i=1}^{d}$.
3: **while** not stopping criterion **do**
4: Compute "decayed error term" $\xi(t)$ for each time stamp t.
5: Compute partial derivatives $\frac{\partial J(\boldsymbol{U},\boldsymbol{W})}{\partial \boldsymbol{U}}$ and $\frac{\partial J(\boldsymbol{U},\boldsymbol{W})}{\partial \boldsymbol{W}^{(i)}}$ using $\xi(t)$ by Eq. (5.14) and Eq. (5.15).
6: Determine the step size λ by line search.
7: Update $\boldsymbol{U} = \boldsymbol{U} - \lambda \frac{\partial J(\boldsymbol{U},\boldsymbol{W})}{\partial \boldsymbol{U}}$.
8: **for** $i = 1, ..., d$ **do**
9: Update $\boldsymbol{W}^{(i)} = \boldsymbol{W}^{(i)} - \lambda \frac{\partial J(\boldsymbol{U},\boldsymbol{W})}{\partial \boldsymbol{W}^{(i)}}$.
10: **end for**
11: **end while**
12: Compute predict adjacency matrix $A(T+1) = \boldsymbol{U}\boldsymbol{V}(T+1)^T = \boldsymbol{U}(\sum_{i=0}^{d} \boldsymbol{W}^{(i)}(T+1)^i)^T$

For undirected networks, the adjacency matrices are symmetric, thus we can leverage symmetric matrix factorization technique [Kuang et al., 2012] to deal with the undirected networks. Then one needs to minimize the loss function $J(\boldsymbol{W})$,

$$J(\boldsymbol{W}) = \sum_{t=1}^{T} \frac{D(t)}{2} \|\mathbf{1}_{E(t)}(\boldsymbol{A}(t) - (\sum_{i=0}^{d} \boldsymbol{W}^{(i)} t^i)(\sum_{i=0}^{d} \boldsymbol{W}^{(i)} t^i)^T)\|_F^2 + \sum_{i=0}^{d} \frac{\beta_i}{2} \|\boldsymbol{W}^{(i)}\|_F^2 \tag{5.16}$$

In order to infer the parameter $\boldsymbol{W}$, we need to compute the derivatives of Eq. (5.16). Similar to the derivative calculation for directed network model, we introduce an "error term" $\psi(t)$ for each time stamp t of undirected networks, as follows:

$$\psi(t) = \mathbf{1}_{E(t)}(\boldsymbol{A}(t) - (\sum_{j=0}^{d} \boldsymbol{W}^{(j)} t^j)(\sum_{j=0}^{d} \boldsymbol{W}^{(j)} t^j)^T) \tag{5.17}$$

Note that, the error matrix in $\psi(t)$ has already projected to indices set defined by $E(t)$. Thus, the derivatives of Eq. (5.16) with regularization term can be calculated as,

$$\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}^{(i)}} = -\sum_{t=1}^{T} D(t)(\psi(t) \sum_{j=0}^{d} \boldsymbol{W}^{(j)} t^j + \psi(t)^T \sum_{j=0}^{d} \boldsymbol{W}^{(j)} t^j) t^i + \beta_i \boldsymbol{W}^{(i)} \tag{5.18}$$

5.1.1.3 Computational Analysis

To help in analyzing the complexity of the algorithm, we assume that the number of nodes in $G(N, \boldsymbol{A}(t))$ is n, the number of edges is m, and the rank of the factorization is k. The gradient-descent method is implemented for M iterations. Here, d is the (polynomial) order of the time-dependent matrix $\boldsymbol{V}(t)$. Furthermore, it is assumed that there are T time-stamps for the temporal method. In each of the M iterations, the bottleneck step involves updating all the parameters.The number of parameters in $\boldsymbol{U}$ is nk. The same number for $\boldsymbol{V}(t)$ is $nk(d+1)$, because we need to sum over the d polynomial orders. for $\{\boldsymbol{W}^{(i)}\}_{i=1}^{d}$, respectively. For the derivative computation of each parameter, one needs to multiply the corresponding k dimensional columns of matrices $\boldsymbol{U}$ and $\sum_{i=0}^{d} \boldsymbol{W}^{(i)} t^i$, and then compute the derivative. This requires time of the order of magnitude of the sum of the corresponding node degrees in the adjacency matrix $\boldsymbol{A}(t)$. By summing up these costs, we obtain:

$$\begin{aligned} & MT \sum_{\{i,j\} \in \boldsymbol{U}} (\textit{degree}_{i,j} + kd) + MT \sum_{\{i,j\} \in \boldsymbol{W}} (\textit{degree}_{i,j} + kd) \\ & = MTmk(d+2) + MTnk^2 d(d+2) \end{aligned} \tag{5.19}$$

The order d of the polynomial is set to a small number such as 1 or 2. Thus, the asymptotic running time is $O(MTmk + MTnk^2)$. Since M, T and k are much smaller than n and m, the time complexity is approximately $O(m+n)$.

5.1.1.4 Leveraging the Factorization in Different Application Settings

The most interesting aspect of the models discussed in this section is its extraordinary *generality* in terms of its applicability to various settings. Most of the existing methods for evolutionary network analysis [Aggarwal and Subbian, 2014] are focused on *specific* problems like link prediction [Sarkar et al., 2012], dynamic community detection [Backstrom et al., 2006, Gupta et al., 2011, Chi et al., 2007, Tang et al., 2008, Mankad and Michailidis, 2013], compression [Sun et al., 2007] and anomaly detection [Ide and Kashima, 2004, Yu et al., 2013a]. Here we provide a very general purpose framework TMF and symmetric TMF (s-TMF), which can perform almost *all* of these tasks within a single unified approach. In the following sections, we will describe how the meth-

ods described in this section can be used to accomplish these tasks.

5.1.2 Temporal Matrix Factorization in Link Prediction

Link prediction and network reconstruction are almost trivial in this setting because the entire network is expressed as a function of time with the use of latent variables. The main advantage of link prediction with temporal matrix factorization model over traditional link prediction methods is that it can not only predict *new links* at any time T in the future, but also predict the *weight of each link*. Note that traditional link prediction is only able to predict new links at "some" point in the future based on the current snapshot, but it does not really provide temporally sensitive analysis. Given the advantages of the proposed model described above, we evaluate the performance of the TMF model and symmetric TMF (s-TMF) model in two aspects, corresponding to link-weight prediction and new link prediction.

To verify the performance of the proposed model, we conducted experiments on a variety of dynamic networks from different domains including three directed networks *UCI Msg*, *Digg* and *Epinions*, and three undirected networks *Infectious*, *arXiv hep-th* and *DBLP*. The detailed descriptions of each network dataset can be found in Section A.1. We consider the canonical link prediction methods as well as recent algorithms as comparative methods, including weighted common neighbors (WCN) [Murata and Moriyasu, 2007], weighted Adamic Adar (WAA) [Zhao et al., 2015], High-performance Link Prediction (HPLP) [Lichtenwalter et al., 2010], Preferential Attachment (PA) [Mitzenmacher, 2004], Nonparametric Link Prediction (NP) [Sarkar et al., 2012], Link Prediction via Matrix Factorization (Fact-Sq) [Menon and Elkan, 2011] and CP Tensor Model (CP-Tensor) [Dunlavy et al., 2011]. Section A.2 has detailed descriptions of each method. Our models are denoted as **TMF** and **s-TMF**, respectively, where the term s stands for symmetric. We attach a suffix (say, d=1), to represent the order of $\boldsymbol{V}(t)$ for the corresponding model.

5.1.2.1 Link Weight Prediction

In this section, we evaluate the link-weight prediction accuracy of each method. For each of the six networks, three directed networks (asymmetric adjacency matrices) and three undirected networks

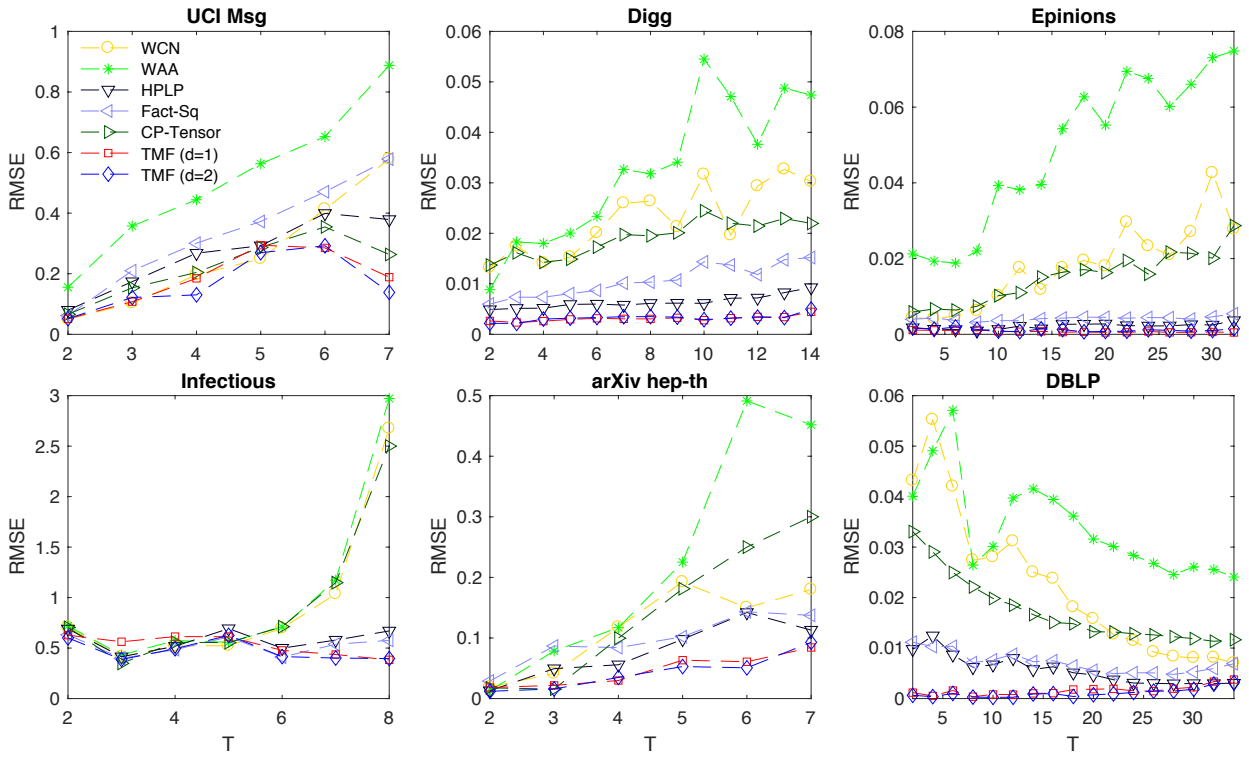


Figure 5.2: Prediction RMSE at time-stamp T

(symmetric adjacency matrices), we choose the first $T-1$ time-stamps as training set, and the T^{th} time-stamp as test set, here T varies from 2 to the total time-stamps of each dataset. We analyze the algorithms by measuring the accuracy of the weight prediction based on root mean-squared error (RMSE). We obtained the RMSE at different T for link prediction as presented in Figure 5.2.

For the first three directed networks *UCI Msg*, *Digg* and *Epinions* in Figure 5.2, we compute the RMSE of all five weighted baselines and our models TMF. Here we set d, the order of the time-dependent matrix, $\boldsymbol{V}(t)$, to 1 (linear) and 2 (quadratic). The other parameter settings are as follows: latent dimension $k = 10$, exponential decay function parameter $\theta = 0.3$, regularizer weights $\alpha = \beta_i = 0.01$. The maximum iteration of TMF models is set to 500. The upper three figures show that the best prediction results are achieved by our TMF models, namely TMF (d=1) and TMF (d=2). The performance of TMF models are very similar with respect to different values of d. The RMSE of TMF models in the last time-stamp of these three directed networks are only 15.58%, 9.49% and 1.34% of the worst baseline, and 36.48%, 48.39% and 26.89% of the best baseline.

For the lower three undirected networks, referred to as *Infectious*, *arXiv hep-th* and *DBLP* in Figure 5.2, we compute the RMSE of all five weighted baselines, TMF and symmetric TMF (s-TMF). Note that we can still use the TMF model for the undirected networks. In this case, $E(t)$ in Eq. (5.6) is the non-zero entries of the upper triangular matrices of the undirected networks. In these detailed figures, we show the results when $d = 1$ (since the results are very similar for $d = 2$) although all results are presented in tabular form in Table 5.1. All the remaining parameter settings are the same. It is evident that TMF and s-TMF provide the best overall performance, which are consistent with the good performance of different values of T in undirected networks. The average RMSE of s-TMF is smaller than TMF, which will be shown in Table 5.1. The RMSE of s-TMF models of these three undirected networks are only 9.49%, 16.78% and 9.13% of the worst baseline, and 42.15%, 67.08% and 73.33% of the best baseline, showing the advantage of TMF and s-TMF models in link-weight prediction.

Table 5.1 displays the average prediction RMSE over all time frames of each dataset. It is evident that TMF and s-TMF models have a lower RMSE than the five weighted baselines. Note that, for undirected networks, s-TMF models always perform better than TMF models, though the RMSE differences are not that large when compare to other baselines. However, the advantage of TMF models is that they can also be applied to directed networks.

Although one might expect the prediction RMSE to decrease with increasing T (and more data), this is not the case for all data sets. It was only in the *UCI Msg* and *Infectious* data sets that the RMSE reduced. For the other four datasets, the RMSE increased. The reason is that the network showed increasing rates of evolution over time with rapid formation of new links. As a result, it became harder to predict more accurately with passage of time. Nevertheless, the incorporation of temporal information still has an inherent temporal advantage over the use of static models; this is the reason that the approach outperforms the baselines.

5.1.2.2 New Link Prediction

The TMF model is not designed for new link prediction since we use Frobenius norm to define the loss function, as shown in Eq. (5.10) and Eq. (5.11). However, one can still apply it to unweighted

Table 5.1: Average RMSE across all time-stamps

Methods	UCI Msg	Digg	Epinions	Infectious	arXiv hep-th	DBLP
WCN	0.2649	0.0229	0.0171	0.9386	0.1167	0.0227
WAA	0.5101	0.0325	0.0483	1.0192	0.2295	0.0350
HPLP	0.2653	0.0064	0.0021	0.5802	0.0785	0.0056
Fact-Sq	0.3324	0.0107	0.0021	0.5215	0.0970	0.0071
CP-Tensor	0.2215	0.0191	0.0150	0.9321	0.1440	0.0170
TMF (d=1)	0.1853	**0.0031**	**0.0007**	0.5304	0.0464	0.0016
TMF (d=2)	**0.1673**	0.0032	0.0012	0.5398	0.0436	0.0014
s-TMF (d=1)	–	–	–	0.4743	0.0432	**0.0011**
s-TMF (d=2)	–	–	–	**0.4391**	**0.0427**	0.0012

data sets. In this section, we predict new links using TMF model on *Epinions* data set in which the maximum link weight is 1.

We measure the accuracy of new link prediction by using the area under curve (AUC) of the receiver operating characteristics (ROC) analysis. The ROC is designed to work in the binary class setting with positive and negative samples. When predicting the new links at time-stamp T, the new edges to be predicted are treated as the "positive" samples. We use the absolute values of our prediction results, so that the new edges (both with $+1$ weights and -1 weights) to be predicted are treated as the positive samples. We then randomly sample the same number of node pairs without an edge between them at time T as "negative" samples ($S(t)$ in Eq. (5.6)). The parameter settings remain the same as previous experiments. We compare TMF (d=1) model with six baselines which is shown in Figure 5.3. In the `Epinions` network, the TMF model outperforms the baseline methods by 15.53% on average. It outperforms *Common Neighbors* by more than 21.55% across all time-stamps. We attribute this success to the temporal nature of the factorization that can predict trends over time, rather than simply relying on a static model.

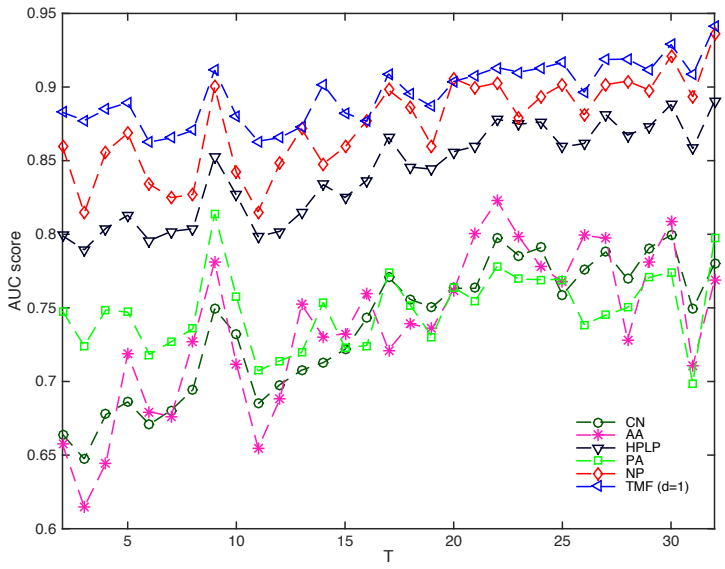


Figure 5.3: New link prediction AUC score at T on *Epinions* dataset

5.1.2.3 Sensitivity Analysis

The loss function defined by Eq. (5.10) is dependent on parameters denoted by k, θ and the regularizer weight α and β_i. Normally, we set the regularizer weight to a relatively small value such as 0.01. Therefore, in this section, we conduct sensitivity analysis on k, the latent dimension of $\boldsymbol{V}(t)$ and θ, the parameter of the exponential decay function $D(t)$.

We used the *Infectious* and *UCI Msg* data sets for sensitivity analysis; the first is a fast evolving undirected network and the second is a slowly evolving directed network. We choose values between k from 5 to 100 for *Infectious* and 10 to 300 for *UCI Msg*. The value of θ varied from 0.1 to 1.0 with interval 0.1. The results are summarized in Figure 5.4. It is evident that RMSE initially improves with k but further improvements are harder beyond a certain point. But from the effectiveness analysis in Section 5.1.1.3 we can see that, the time complexity is proportion to $O(k^2)$ for fixed networks. Taking both prediction error and computational time into consideration, we will choose a relative small k from 10 to 100.

The value of θ has a significant impact on prediction results, and it is in fact sensitive to the data set. For the *Infectious* dataset, when θ increases, the proposed model will have a better RMSE. This means that as we assign less weight on the early time-stamps, it will achieve better prediction

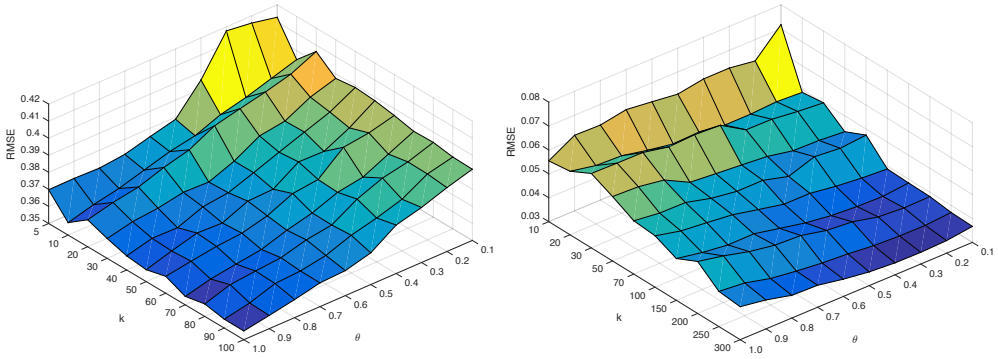


Figure 5.4: Last time-stamp prediction RMSE of *Infectious* and *UCI Msg*

results. Thus, we can infer that for a fast evolving network, higher exponential weight decay brings better prediction accuracy. For *UCI Msg* dataset, we obtain a totally different trend. As θ increases, RMSE increases. This means that for a slowly evolving network like *UCI Msg*, we should assign more weights to the early time-stamps.

5.1.3 Temporal Matrix Factorization in Anomaly Detection

Temporal anomalies and events are often defined by *unexpected* changes in the network structure over time, which are different from their forecasted values. Consider the scenario, where the training data at time stamps $1 \ldots T$ has been used to learn the network $\boldsymbol{A}(t)$. We would like to use the learned training data to determine the anomalies in $\boldsymbol{A}(T+1)$, when the network at time $(T+1)$ is received. Let the predicted network at time $(T+1)$ according to the aforementioned model be denoted by $\hat{\boldsymbol{A}}(T+1)$, and the true network state at time $(T+1)$ be $\boldsymbol{A}(T+1)$. Then, the *unexpected* part of the change in network structure, is given by the following:

$$\Delta \boldsymbol{A}(T+1) = \hat{\boldsymbol{A}}(T+1) - \boldsymbol{A}(T+1) \tag{5.20}$$

Large *absolute* values in $\Delta \boldsymbol{A}(T+1)$ correspond to edges with unusual levels of activity:

$$F(t) = \{(i,j) : |[\Delta \boldsymbol{A}(T+1)]_{ij}| > \delta\} \tag{5.21}$$

One can use a Z-statistic or t-statistic over the values of $|\Delta \boldsymbol{A}(T+1)|$ to determine the value of δ at which the change is significant. As in the case of communities, one can discover the connected components in such edge sets, and report the anomalous change regions in the network. It is also

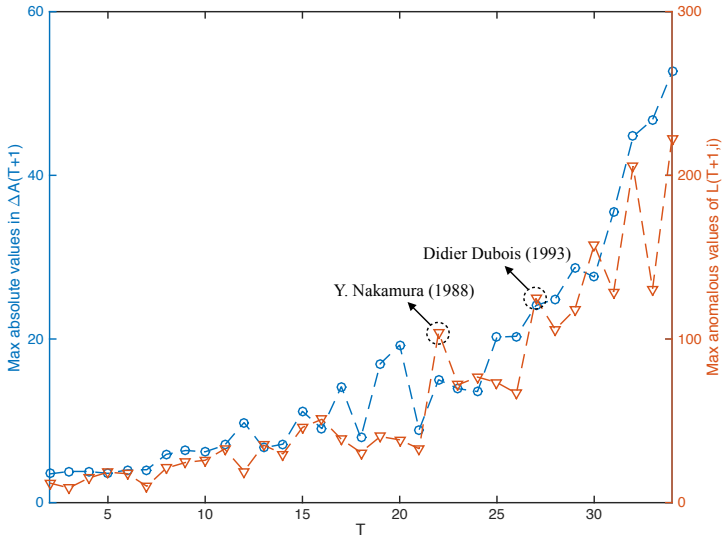


Figure 5.5: Maximum value distributions of $\Delta A(T+1)$ and $L(T+1,i)$ across all time-stamps

possible to discover the individual node hot spots, by first quantifying the level $L(T+1,i)$ of anomalous activity adjacent to each node i at time $(T+1)$ as follows:

$$L(T+1,i) = \sum_{j:\{i,j\}\in F(t)} |[\Delta \boldsymbol{A}(T+1)]_{ij}| \tag{5.22}$$

Nodes with large anomalous values of $L(T+1,i)$ correspond to hot-spots of activity. One can use a Z-statistic or t-statistic to determine thresholds on the value of $L(T+1,i)$. For example, in a DBLP network, such nodes might correspond to researchers who had a sudden change in their co-authorship activity as a result of an event, such as change of institution. Therefore, the structural and node events tell us a lot about unusual events in the underlying network activity.

5.1.3.1 Discovering Temporal Anomalies: A Case Study

In this section, we detect temporal anomalous coauthors and individual authors on DBLP dataset using TMF model. Based on aforementioned analysis, we first show the trends of maximum absolute values in $\Delta \boldsymbol{A}(T+1)$ and maximum anomalous values of level $L(T+1,i)$. Then we present the top 10 pairs of anomalous coauthors and authors.

The blue dashed line in Figure 5.5 shows the maximum absolute value in $\Delta \boldsymbol{A}(T+1)$ at each time-stamp T, while the red dashed line shows the maximum anomalous value of $L(T+1,i)$.

Table 5.2: Top 10 pairs of anomalous coauthors sorted by $|[\Delta \boldsymbol{A}(T+1)]_{ij}|$

#	Δ	Coauthors	Year
1	52.6421	Sudhakar M. Reddy, Irith Pomeranz	1999-2000
2	46.8029	Sudhakar M. Reddy, Irith Pomeranz	1998-1999
3	44.7452	Sudhakar M. Reddy, Irith Pomeranz	1997-1998
4	36.8849	Raj Jain, Sonia Fahmy	1997-1998
5	36.877	Raj Jain, Rohit Goyal	1997-1998
6	35.5676	Sudhakar M. Reddy, Irith Pomeranz	1996-1997
7	35.2318	Divyakant Agrawal, Amr El Abbadi	1999-2000
8	34.876	Sonia Fahmy, Rohit Goyal	1997-1998
9	32.2694	Divyakant Agrawal, Amr El Abbadi	1998-1999
10	30.5123	Didier Dubois, Henri Prade	1997-1998

Therefore, these two lines respectively represent the trends of top anomalous coauthor and top anomalous author at each time-stamp. It can be seen that as T increases, the anomalous level increases in terms of both quantifications. This can be explained by the fact that recent years have experienced a larger volume of publications; therefore, the corresponding anomalous trends increase in absolute magnitude. Furthermore, the statistical correlation between these two lines is very high (0.9308), which is evidence that anomalous author-events are usually caused by underlying anomalous co-authorship events.

Additionally, we labeled two anomalous peaks, corresponding to *Y. Nakamura* (1988) and *Didier Dubois* (1993) in Figure 5.5. These two authors represent different types of abnormalities. According to *DBLP*, *Y. Nakamura* has only one publication (in 1982) before 1988 but has 15 in 1988, which is unusual. For *Didier Dubois*, the abnormality is a result of the fact that he has an unusually large number of new coauthors in 1993.

We then detect the anomalous coauthors by calculating the unexpected part of the change $\Delta \boldsymbol{A}(T+1)$ in network structure. Table 5.2 represents the top 10 pairs of anomalous coauthors sorted by $|[\Delta \boldsymbol{A}(T+1)]_{ij}|$. The experiment is conducted with the same parameter settings de-

Table 5.3: Top 10 anomalous authors sorted by the level of anomalous activity $L(T+1,i)$

#	$L(T+1,i)$	Author	Year
1	222.799	Robin J. Chapman	1999-2000
2	205.561	Hector Garcia-Molina	1997-1998
3	190.993	John H. Lindsey II	1999-2000
4	168.677	Raj Jain	1997-1998
5	157.488	Alberto L. S.-V.	1995-1996
6	152.147	David Callan	1999-2000
7	149.377	Rohit Goyal	1997-1998
8	148.404	Sonia Fahmy	1997-1998
9	147.063	Hugo De Man	1999-2000
10	144.791	Alberto L. S.-V.	1999-2000

scribed in Section 5.1.2.1. Surprisingly, the pair of authors *Sudhakar M. Reddy* and *Irith Pomeranz* appears 4 times. The total coauthor papers between them from 1997 to 2000 are 92, 116, 135 and 159, respectively. But before 1991, they have no coauthor papers. This could be the reason that why they are titled "top anomalous coauthor" by our model.

Additionally, we calculate $L(T+1,i)$ of all time-stamps and show the top 10 anomalous values in Table 5.3. We can verify the authors who receive large anomalous values with `DBLP` database. For example, the *DBLP* homepage shows that *Robin J. Chapman*, the top 1 "anomalous" author, had no coauthor paper in 1999, but 21 coauthor papers in 2000[1].

5.1.4 Temporal Matrix Factorization in Community Analysis

The temporal matrix factorization approach can be naturally used for discovering the rate of evolution of each edge in the network at any given time t. This may be expressed in the form of the

[1] `http://dblp.uni-trier.de/pers/hd/c/Chapman:Robin_J=`

following matrix $\boldsymbol{A}'(t)$ at any given time t:

$$\boldsymbol{A}'(t) = \frac{\partial \boldsymbol{A}(t)}{\partial t} = \frac{\partial f(\boldsymbol{U}\boldsymbol{V}(t)^T)}{\partial t} \tag{5.23}$$

Here we only consider the temporal matrix factorization model for directed network, since it is trivially generalizable to undirected network. Note that $\boldsymbol{A}'(t)$ can vary with t, when a nonlinear expression is used for $f(\boldsymbol{U}\boldsymbol{V}(t)^T)$. When a polynomial expression of $\boldsymbol{V}(t)$ is used, the above partial derivative evaluates to

$$\frac{\partial f(\boldsymbol{U}\boldsymbol{V}(t)^T)}{\partial t} = \boldsymbol{U}(\sum_{i=1}^{d} \boldsymbol{W}^{(i)} t^{i-1} i)^T) \tag{5.24}$$

The partial derivative of Eq. (5.24) is equal to $\boldsymbol{U}\boldsymbol{W}^{(1)}$ when $\boldsymbol{V}(t)$ is linear ($i = 1$). Note that edges (i, j) for which $[\boldsymbol{A}'(t)]_{ij} = \sum_{p=1}^{k}(u_{ip} \sum_{q=1}^{d} w_{jp}^{(q)} t^{q-1} q) > 0$ correspond to edges in which the weights are increasing with time. When the sign is negative, it corresponds to edges with reducing weights. One approach to discover expanding communities, is to isolate the following edge set for some threshold $\delta > 0$:

$$I(t) = \{(i, j) : [\boldsymbol{A}'(t)]_{ij} > \delta\} \tag{5.25}$$

The connected components of this network yield the expanding communities. By varying the value of δ, one can obtain expanding communities at different threshold levels of evolution. A similar approach can be used to determine the contracting communities. It is noteworthy that the discovery of expanding or contracting communities is almost trivial once the network has been expressed in functional form.

5.1.4.1 Discovering Expanding Communities: A Case Study

Expanding and contracting community detection techniques are essential for finding trends in time-series data, such as the discovery of hot research topics. In this section, we report the expanding community detection examples from the *DBLP* data set using the TMF model.

In order to interpret each expanding community detected by the proposed model, we label the coauthor edges with their *dominant* venues (journals or conferences). For example, if author a_1 and a_2 coauthored 10 papers (7 in SIGIR, 2 in WWW and 1 in WSDM), we label the coauthor

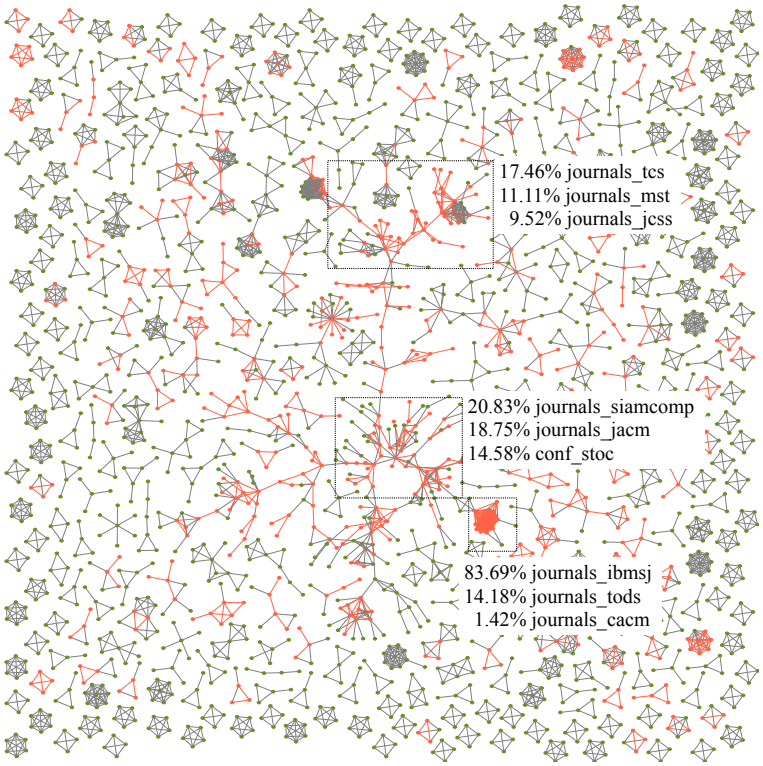


Figure 5.6: Expanding communities from 1980 to 1981

edge e_{a_1,a_2} with SIGIR. We utilize asymmetric TMF model with linear $\boldsymbol{V}(t)$ in this experiment. The gradient threshold δ is set to 0.01. We filter out the graph noise by building a set of trees using breadth-first search (BFS) over the entire vertex set of the network and then considering only those trees whose vertex set size are at least 4. The other parameter settings remain the same as Section 5.1.2.1. The expanding communities are shown in Figure 5.6.

Figure 5.6 illustrates three largest expanding communities in *DBLP* from 1980 to 1981. The top three venues of each community are also listed. It can be seen that, the largest expanding community has 17.46%, 11.11% and 9.52% of the weight increasing edges labeled with *journals_tcs* (Theoretical Computer Science), *journals_mst* (Mathematical Systems Theory) and *journals_jcss* (Journal of Computer and System Sciences). Notably, these three venues belong to the same area *Computer Science Theory* which is consistent with the increasing prominence of theoretical computer science in the early eighties. Similar analytical results were obtained from the other two expanding communities. Among the edges with increasing weight, the second community has 20.83%, 18.75% and 14.58% of the edges labeled with *journals_siamcomp* (SIAM Journal

on Computing), *journals_jacm* (Journal of the ACM) and *conf_stoc* (Symposium on Theory of Computing). These venues also tend to contain theoretical topics. The third community is a highly compact sub-network where 83.69% edges are labeled with *journals_ibmsj* (IBM Systems Journal). Another interesting finding from Figure 5.6 is that most of the cliques contain fewer edges with increasing weight. This means that stable groups of persistent co-authors often have a tendency to not evolve too much with time. This is consistent with the intuition that most evolution in network structures is caused by dynamically changing co-authorships. On the other hand, connected components with a greater number of branches also tend to have more edges with increasing weight. This is because these components contain groups of authors that are more open to initiating new cross-collaborations with different groups. In these sense evolutionary community detection can provide insights about the *causality* of the underlying network changes because of its summary representation.

5.2 Modeling Co-Evolution Across Multiple Networks

Evolving networks are common in a wide variety of settings because of the importance of different types of dynamic networks and social streams [Aggarwal and Subbian, 2014]. In many cases, multiple networks co-evolve with time and influence each other's co-evolution. The co-evolution of multiple networks often has significant predictive power in the context of a wide variety of applications. In this section, we will examine the problem of co-evolution in the context of networks that are defined on the same set of nodes, but multiple networks are defined over edges with different types of interpretations. Some examples of such networks are as follows:

- The same author may be represented in multiple collaboration networks such as patent networks and publication networks. Each of these areas forms a collaboration network of which nodes are authors and edges represent co-authorships between two authors. The activity in one network is typically related to that in other networks.

- In social networks, users may belong to different types of implicitly constructed networks. For example, users may exchange posts, post a "like" on each other's posts, or interact with

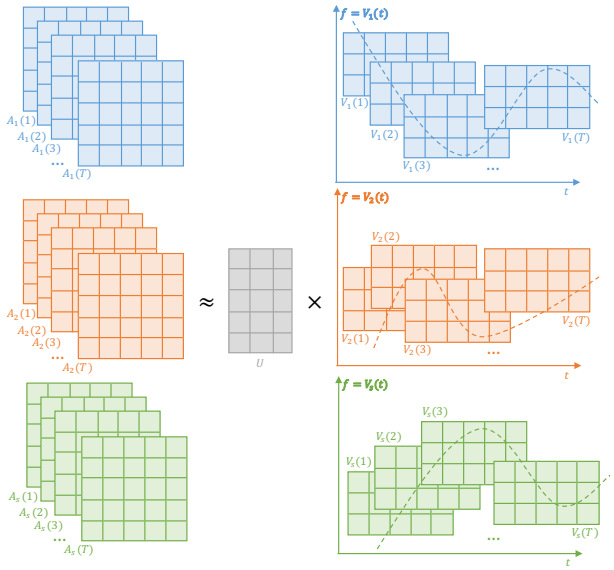


Figure 5.7: Illustration of shared temporal matrix factorization at time-stamp t

the same object. Each of these different types of interactions can be used to create a different type of network in its own right. Furthermore, interactions among different networks could affect each other.

- In some cases, a set of participants may be connected in different ways across multiple networks. For example, they may be connected via both communication and social network links. The interaction dynamics in one network can often be predictive of the interaction dynamics in another network.

Co-evolving networks, which are defined on the same set of nodes, can also be viewed as heterogeneous information networks, in which links of a specific type can be used to define an individual (homogeneous) network. However, here we will treat each such homogenous network as an individual entity, which co-evolves with other homogeneous networks over time.

The prediction of co-evolution in networks is often challenging because of its structural nature. A network may contain large parts that evolve with time, and the changes in the structure of various edges might be correlated within particular structural localities. This cannot be easily captured with mathematical models. To address these issues, we propose to use a *shared temporal matrix factorization* framework CoEvol, in which some of the factors are shared between networks.

Fig. 5.7 shows the shared temporal matrix factorization at time-stamp t. Formally, the adjacency matrices $\{\boldsymbol{A}_r(t)\}_{r=1}^s$ are decomposed into a product of $\boldsymbol{U}$ and $\{\boldsymbol{V}_r(t)\}_{r=1}^s$. Here $\boldsymbol{U}$ is a constant matrix shared by all co-evolving networks. $\{\boldsymbol{V}_r(t)\}_{r=1}^s$ are time-dependent matrices corresponding to individual networks. s is the total number of networks in our setting. The shared factorization process has the advantage that it converts a sequence of networks to sets of real-valued time-series. Real-valued time-series are much easier to analyze in terms of auto-correlations, cross-correlations, and lag-correlations. Furthermore, one can also use the approach to discover co-evolving communities and predict temporal structure in the multiple networks.

5.2.1 The Network Co-Evolution Model

In this section, we will introduce the network co-evolution model as a function of time. We assume that we have s different networks $\{G_1(t) = (N, \boldsymbol{A}_1(t)) \ldots G_s(t) = (N, \boldsymbol{A}_s(t))\}$ that are defined over the same set of nodes and co-evolve over time. It is assumed that the $(i, j)^{th}$ entry of the adjacency matrix of $\boldsymbol{A}_r(t) \in \mathbb{R}^{n \times n}$ is given by $a_{ijt}^{(r)}$, where $n = |N|$. The adjacency matrix might either be binary or weighted. Our approach is capable of handling both settings. Furthermore, the networks might be either directed or undirected, and our approach is capable of handling both settings. The time-stamp t is a discrete and ordered variable that varies from 1 to T. Furthermore, we assume that N is a fixed set of nodes corresponding to the union of all nodes received till time T, over which all networks are defined.

In this study, we propose to use *shared temporal factorization methods* for performing evolutionary network analysis. The basic idea is to use a set of constant factor matrix $\boldsymbol{U} \in \mathbb{R}^{n \times k}$ and a temporally varying set of factors $\boldsymbol{V}_r(t) \in \mathbb{R}^{k \times n}$ for the r^{th} network $G_r(t)$, where k is the latent dimension. Note that the temporal factors $\boldsymbol{V}_r(t)$ are functions defined over the entire timestamps

that are known to us. Therefore, we have the following:

$$\boldsymbol{A}_1(t) = \boldsymbol{U}\boldsymbol{V}_1(t)^T$$
$$\boldsymbol{A}_2(t) = \boldsymbol{U}\boldsymbol{V}_2(t)^T$$
$$\ldots$$
$$\boldsymbol{A}_s(t) = \boldsymbol{U}\boldsymbol{V}_s(t)^T$$

In addition, we impose a non-negativity constraint on the factors for greater interpretability (see Section 5.2.1.2). Thus $\boldsymbol{U}$ and $\boldsymbol{V}_r(t)$ have non-negative entries. According to the theory that learns the parts of objects by non-negative matrix factorization [Lee and Seung, 1999], the k columns of $\boldsymbol{U}$ are called basis communities (or *aspects*). Each column of $\boldsymbol{V}_r(t)^T$ is called an encoding which is in one-to-one correspondence with an adjacency vector in the matrix $\boldsymbol{A}_r(t)$. An encoding consists of the coefficients by which the weights of all edges associate to a node is represented with a linear combination of basis aspects. The functions $\boldsymbol{V}_r(t)$ can take on any canonical form in terms of variation with t, such as linear, polynomial, and so on.

There is no algorithm known yet for computing an exact non-negative matrix factorization despite its existence [Vasiloglou et al., 2009]. Therefore we would like to determine the values of $\boldsymbol{U}$ and $\boldsymbol{V}_r(t)$ so that each $\boldsymbol{A}_r(t)$ matches $\boldsymbol{U}\boldsymbol{V}_r(t)^T$ as closely as possible. Although the standard approach in matrix factorization is to optimize the Frobenius error of the factorization, we cannot separately optimize the Frobenius norm of $\boldsymbol{A}_r(t) - \boldsymbol{U}\boldsymbol{V}_r(t)^T$ because the matrix $\boldsymbol{U}$ is shared across the s different factorizations. Therefore, we optimize the sum of squared errors across the s different factorizations. Note that instead of minimizing the Frobenius error to determine suitable factor matrices, we could also minimize general divergence measures. But this would not make much of a difference [Lee and Seung, 2001]. In addition, we would like to compute the error over all values of t from 1 to the current time-stamp T. A time-decay function $D(t)$ is used to regulate the varying importance of different time-stamps:

$$\min_{\boldsymbol{U},\boldsymbol{V}_r \geq 0} J = \sum_{t=1}^{T} \frac{D(t)}{2} \sum_{r=1}^{s} \|\boldsymbol{A}_r(t) - \boldsymbol{U}\boldsymbol{V}_r(t)^T\|_F^2 \tag{5.26}$$

Here, $D(t)$ is a decay function with time t that regulates the greater importance of the current state of the network with respect to the past time-stamps. For example, one might choose the decay

function as the exponential decay function $D(t) = e^{-\theta(T-t)}$ with parameter $\theta > 0$.

Most real networks are sparse, as a result of which only a small fraction of the edges are present in $\boldsymbol{A}_r(t)$. When the number of nodes is large, the entire objective function requires $O(n^2)$ time to evaluate. This can be inefficient because of the entries contain values of 0. Therefore, we will use all the non-zero entries in $\boldsymbol{A}_r(t)$ along with a sample of 0 entries from $\boldsymbol{A}(t)$ [Aggarwal, 2016]. Let $S_r(t)$ be a sample of edges (i, j) at time-stamp t such that the value of $a_{ijt}^{(r)}$ is 0. At time t, let $E_r(t)$ be the set of edges for which the weights in $\boldsymbol{A}(t)$ are non-zero at time t, along with the sample of 0 edge weights. Therefore, we have the following:

$$E_r(t) = \{(i, j)|a_{ijt}^{(r)} > 0\} \cup S_r(t) \tag{5.27}$$

Then, the aforementioned objective function can be written in terms of the specified entries as follows:

$$J(\boldsymbol{U}, \boldsymbol{V}) = \sum_{t=1}^{T} \frac{D(t)}{2} \sum_{r=1}^{s} \sum_{(i,j)\in E_r(t)} (a_{ijt}^{(r)} - (\boldsymbol{U}\boldsymbol{V}_r(t)^T)_{ij})^2 \tag{5.28}$$

Note that the model described in the above objective function can still be used for the undirected co-evolving networks. But we can also decompose the symmetric adjacency matrices of undirected networks into a product of $\boldsymbol{V}_r(t)$ and its transpose for each network, and leverage symmetric matrix factorization technique [Kuang et al., 2012] to find the solution.

5.2.1.1 Solving the Optimization Problem

The function $\boldsymbol{V}_r(t)^T$ can take on any canonical form based on the specific tasks. One typical choice for $\boldsymbol{V}_r(t)^T$ is the linear function [Yu et al., 2017c], that is $\boldsymbol{V}_r(t) = \boldsymbol{X}_r t + \boldsymbol{Y}_r$. Since $\boldsymbol{V}_r(t) \geq 0$, here we attempt to use a more stringent constraint: $\boldsymbol{X}_r \geq 0$ and $\boldsymbol{Y}_r \geq 0$. Given the definition of $\boldsymbol{V}(t)$, the matrix form of Eq. (5.28) becomes the following with added regularization:

$$\begin{aligned} J(\boldsymbol{U}, \boldsymbol{X}, \boldsymbol{Y}) = &\sum_{t=1}^{T} \frac{D(t)}{2} \sum_{r=1}^{s} \|\mathbf{1}_{E(t)}(\boldsymbol{A}_r(t) - \boldsymbol{U}(\boldsymbol{X}_r t + \boldsymbol{Y}_r)^T)\|_F^2 \\ &+ \frac{\alpha}{2}\|\boldsymbol{U}\|_F^2 + \frac{\beta}{2}\|\boldsymbol{X}\|_F^2 + \frac{\gamma}{2}\|\boldsymbol{Y}\|_F^2 \end{aligned} \tag{5.29}$$

where,

$$\mathbf{1}_{E(t)}(\boldsymbol{W}) = \begin{cases} \boldsymbol{W}_{ij} & \text{if } (i,j) \in E(t), \\ 0 & \text{if } (i,j) \notin E(t). \end{cases}$$

Then the bound-constrained non-negative matrix factorization optimization problem becomes,

$$\begin{aligned} \min_{\boldsymbol{U},\boldsymbol{X},\boldsymbol{Y}} \quad & J(\boldsymbol{U},\boldsymbol{X},\boldsymbol{Y}) \\ \text{subject to} \quad & \boldsymbol{U} \geq 0, \boldsymbol{X} \geq 0, \boldsymbol{Y} \geq 0 \end{aligned} \tag{5.30}$$

For the ease of the gradient calculation, we introduce a "error term" $\xi(t,r)$ for each network r at time stamp t, as follows:

$$\xi(t,r) = \mathbf{1}_{E(t)}(\boldsymbol{A}_r(t) - \boldsymbol{U}(\boldsymbol{X}_r t + \boldsymbol{Y}_r)^T) \tag{5.31}$$

To compute the gradient, we will need to differentiate our error function. Since our function is defined by parameter matrices $\boldsymbol{U}$, $\boldsymbol{X}$ and $\boldsymbol{Y}$, we will need to compute a partial derivative for each. These derivatives work out to be:

$$\frac{\partial J(\boldsymbol{U},\boldsymbol{X},\boldsymbol{Y})}{\partial \boldsymbol{U}} = \sum_{t=1}^{T} D(t) \sum_{r=1}^{s} \xi(t,r)(-\boldsymbol{X}_r t - \boldsymbol{Y}_r) + \alpha \boldsymbol{U} \tag{5.32}$$

$$\frac{\partial J(\boldsymbol{U},\boldsymbol{X},\boldsymbol{Y})}{\partial \boldsymbol{X}_r} = \sum_{t=1}^{T} D(t) \sum_{r=1}^{s} \xi(t,r)^T(-\boldsymbol{U}t) + \beta \boldsymbol{X}_r \tag{5.33}$$

$$\frac{\partial J(\boldsymbol{U},\boldsymbol{X},\boldsymbol{Y})}{\partial \boldsymbol{Y}_r} = \sum_{t=1}^{T} D(t) \sum_{r=1}^{s} \xi(t,r)^T(-\boldsymbol{U}) + \gamma \boldsymbol{Y}_r \tag{5.34}$$

We now have all the derivatives needed to run gradient descent. Pseudocode of the full approach is given in Algorithm 5, where λ is the learning rate. To train this model, we can now repeatedly take steps of gradient descent to reduce our cost function $J(\boldsymbol{U},\boldsymbol{X},\boldsymbol{Y})$, here we use the projected gradient methods [**?**]. The iterative algorithm starts from non-negative initial conditions for $\boldsymbol{U}$, $\boldsymbol{X}$ and $\boldsymbol{Y}$, iteration of these update rules for non-negative $\boldsymbol{A}_r(t)$ finds an approximate factorization $\boldsymbol{A}_r(t) \approx \boldsymbol{U}\boldsymbol{V}_r(t)^T$ for each network by converging to a local minimum of the objective function given in Eq. (5.29). Monotonic convergence can be proven using techniques similar to those used in proving the convergence of the EM algorithm [Lee and Seung, 1999].

Algorithm 5 Projected Gradient for The COEVOL Model

Input: Adjacency matrices $\{\boldsymbol{A}_1(t), \boldsymbol{A}_2(t), ..., \boldsymbol{A}_s(t)\}_{t=1}^{T}$, and the latent dimension k.

Output: Results of factor matrices $\boldsymbol{U}$ and $\boldsymbol{V}_r(t)$, $r = 1, ..., s$ for each time-stamp.

Set latent dimension k.
Randomly initialize $\boldsymbol{U} \geq 0$, $\boldsymbol{X} \geq 0$ and $\boldsymbol{Y} \geq 0$.
while not stopping criterion **do**
    Compute "decayed error term" $\xi(t, r)$ for network r at each time-stamp t.
    Compute partial derivatives $\frac{\partial J(\boldsymbol{U},\boldsymbol{X},\boldsymbol{Y})}{\partial \boldsymbol{U}}$ using Eq. (5.32).
    Compute partial derivatives $\frac{\partial J(\boldsymbol{U},\boldsymbol{X},\boldsymbol{Y})}{\partial \boldsymbol{X}_r}$ using Eq. (5.33).
    Compute partial derivatives $\frac{\partial J(\boldsymbol{U},\boldsymbol{X},\boldsymbol{Y})}{\partial \boldsymbol{Y}_r}$ using Eq. (5.34).
    Determine the step size λ by line search.
    Update $\boldsymbol{U} = \boldsymbol{U} - \lambda \frac{\partial J(\boldsymbol{U},\boldsymbol{X},\boldsymbol{Y})}{\partial \boldsymbol{U}}$.
    for $r = 1, ..., s$ **do**
        Update $\boldsymbol{X}_r = \boldsymbol{X}_r - \lambda \frac{\partial J(\boldsymbol{U},\boldsymbol{X},\boldsymbol{Y})}{\partial \boldsymbol{X}_r}$.
        Update $\boldsymbol{Y}_r = \boldsymbol{Y}_r - \lambda \frac{\partial J(\boldsymbol{U},\boldsymbol{X},\boldsymbol{Y})}{\partial \boldsymbol{Y}_r}$.
    end for
    $\boldsymbol{X}_r = max\{\boldsymbol{0}, \boldsymbol{X}_r\}$, $\boldsymbol{Y}_r = max\{\boldsymbol{0}, \boldsymbol{Y}_r\}$
end while
Compute time-dependent factor matrix $\boldsymbol{V}_r(t) = \boldsymbol{X}_r t + \boldsymbol{Y}_r$, $r = 1, ..., s$.

5.2.1.2 Interpretation of the COEVOL Model

The main advantage of introducing a non-negativity constraint is that a high degree of interpretability can be achieved in the COEVOL model. Take Fig. 5.7 as an example: assume that there are $s = 3$ networks, each of which is denoted as a toy 5×5 matrix. The rank of the factorization, also referred to as the number of *aspects*, is 3 in this example. Since matrices $\{\boldsymbol{A}_r(t)\}_{r=1}^{s}$ are directed graphs, the element $a_{ijt}^{(r)}$ is the weight that measures the interaction between sender node n_i and receiver node n_j of network r at timestamp t. Therefore, the corresponding factor matrices provide a clear interpretability about the affinity of sender nodes and receiver nodes to these aspects. If we use collaboration networks in *DBLP* as a concrete illustration, the three networks could be *data*

mining network, machine learning network and *database network*, and the aspects are the general topics in these networks such as *supervised learning, sequential data analysis* and *graph analysis*. The entry values in matrices $\boldsymbol{U}$ and $\boldsymbol{V}_r(t)$ quantify the level of interest in the relevant aspect. Notably, the COEVOL framework is trivially generalizable to undirected networks, since undirected networks are special cases of directed networks.

5.2.1.3 Computational Analysis

We assume that the number of networks is s, the number of nodes in $G_s(t)$ is n, the number of edges is m, and the rank of the factorization is k. The iteration threshold for Algorithm 1 is set as M. Furthermore, we assume that the total number of time-stamps is T. In each of the M iterations, the number of parameters in $\boldsymbol{U}$ is nk. The number of parameters in $\boldsymbol{X}_r$ and $\boldsymbol{Y}_r$ is also nk for each network r. For the derivative computation of each parameter at time-stamp t, one needs to calculate the "error term" $\xi(t, r)$. This requires time $O(mk)$. Since we have $(2s+1)nk$ parameters, the total cost will be $MT \cdot (2s+1)nk \cdot mk$. Thus the asymptotic running time is $O(MTsmnk^2)$.

5.2.1.4 Applications

Numerous applications can be designed in the context of such co-evolving time-series because the approach transforms temporal co-evolving networks to co-evolving time-series with real-values. In the following sections, we will first deploy the COEVOL framework to cross-network structure prediction task. Then we detect the lag correlations among the co-evolving time-series. The evolutions of the time-series among the different groups may be correlated with one another. We show the improvements in structure prediction task with the help of the lag correlations. Additionally, the cross network communities can be detected by clustering the multidimensional data corresponding to the rows of the shared factor matrix $\boldsymbol{U}$. We evaluate the detected communities using cluster purity.

Table 5.4: Conferences in each network dataset

Network	Conferences
Data Mining	KDD, ICDM, CIKM, SDM, PKDD, WSDM
Machine Learning	ICML, NIPS, IJCAI, AAAI, UAI, ECML
Database	VLDB, ICDE, SIGMOD, PODS, EDBT, ICDT

5.2.2 Cross-Network Link Prediction

In this section we use the co-evolution information among multiple networks to help predict the entire network structure at each timestamp. The ability of this share temporal factorization model to predict links at a specific time is so general, that one can effectively reconstruct network structure at any given time. In this section, we evaluate the link prediction performance of CoEvol against different baselines.

5.2.2.1 Co-Evolving Network Datasets and Baselines

We experimentally evaluated the performance of our CoEvol framework on a variety of dynamic networks obtained from the *DBLP* database, which provides co-authorship information for major computer science journals and conferences. In this paper, the data mining network (*DM*), machine learning network (*ML*) and databases network (*DB*) are built based on *DBLP* database. Table 5.4 summarizes the conferences that included in each network.

We also use both conference network (*Conf*) and journal network (*Journal*) from *DBLP* database in which edges denote the co-authorship from conferences or journals. The nodes in these networks represent the authors, and edge weights represent the number of co-authorships between two authors. Since authors can involve in various networks, we create our test datasets by combining different networks using the union set of the nodes and edges. The statistics of all four datasets *DM|ML*, *DM|DB*, *DM|ML|DB* and *Conf|Journal* are shown in Table 5.5.

For comparison, we consider the following link prediction methods: weighted common neighbors (WCN) [Zhao et al., 2015], weighted Adamic Adar (WAA) [Adamic and Adar, 2003], High-

Table 5.5: Co-evolving network dataset description

Dataset	Vertices	Unique Edges	Max Weight	$\|T\|$	#Networks
DM|ML	35,935	49,746 | 53,132	28 | 35	24 years	2
DM|DB	27,782	49,746 | 42,895	28 | 56	24 years	2
DM|ML|DB	44,293	49,746 | 53,132 | 42,895	28 | 35 | 56	24 years	3
Conf|Journal	315,979	503,305 | 382,099	117 | 80	34 years	2

performance Link Prediction (HPLP) [Lichtenwalter et al., 2010], Non-negative Matrix Factorization (NMF) [Lee and Seung, 1999] and CP Tensor Model (CP-Tensor) [Dunlavy et al., 2011]. Note that the *HPLP* algorithm used here is a modified version that trains a regression model to predict the link weights. The evaluation metric we adopt is the root mean-squared error (RMSE).

5.2.2.2 Experimental Results

In this section, we present an experimental comparison of our model COEVOL with the link prediction baselines. For each of the four co-evolving network datasets, we evaluate the link-weight prediction accuracy across all time-stamps. In detail, the first $t-1$ time-stamps of each dataset are used as the training set, and the t^{th} time-stamp is used as the testing set ($t \in [2, T]$). We analyze the baselines by measuring the link weight prediction accuracy based on the root mean-squared error (RMSE). The RMSE at different t for link prediction is presented in Fig. 5.8.

For these four co-evolving network datasets *DM|ML*, *DM|DB*, *DM|ML|DB* and *Conf|Journal*, we compute the prediction RMSE of all five baselines and our model COEVOL. The latent dimension k of factors $\boldsymbol{U}$, $\boldsymbol{X}_r$ and $\boldsymbol{Y}_r$ is set to 10. The other model parameter settings are as follows: exponential decay function parameter $\theta = 0.3$, regularizer weights $\alpha = \beta = \gamma = 0.01$. The maximum iteration of COEVOL model is set to 500. We set the same latent dimension k for baseline methods NMF and CP-Tensor. The results in Fig. 5.8 demonstrate that the best prediction results are achieved by our COEVOL model. It is evident that COEVOL provides the best overall performance, which are consistent across all time-stamps. The RMSE of COEVOL in the last time-stamp is only 35.27%, 36.75%, 49.29% and 30.06% of that of the worst baseline, and 69.35%,

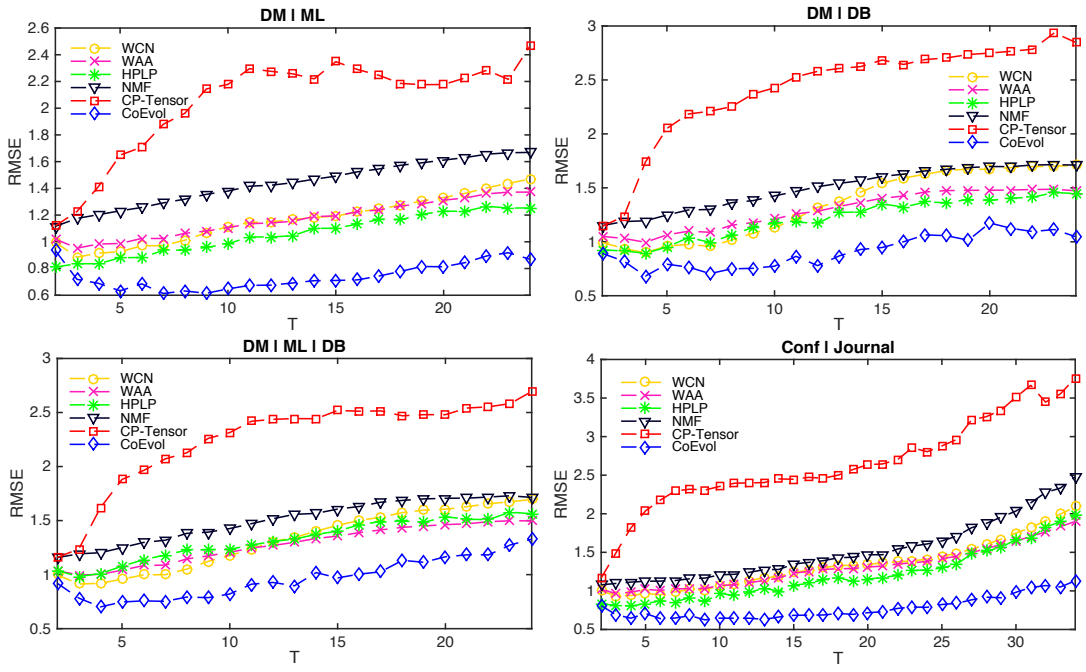


Figure 5.8: Prediction RMSE at time-stamp t, $t \in [2, T]$

72.31%, 88.70% and 59.50% of that of the best baseline of these four datasets, respectively. The result shows that our model can better capture the underlying structure of the network, and also have a better prediction ability. This is due to the time-dependent factors $\boldsymbol{V}_r(t)$ that enable the model to better fit the data across different time-stamps. Surprisingly, the performance of CP-Tensor is much worse among all baselines. This method is implemented based on the heuristic approach [Dunlavy et al., 2011], which assume that average activity in the last three time-stamps is a good choice for the weight.

The average prediction RMSE across all time-stamps of each dataset are summarized in Table 5.6. Methods with scores in bold have the best score among methods being compared in that dataset. The COEVOL model outperforms other baselines by a large margin. WCN and WAA perform well due to the normalization by the size of common neighbors. We tried the original version of WCN and WAA, their RMSE are worse than CP-Tensor if not normalized.

Table 5.6: Average RMSE across all time-stamps on four network datasets

Methods	*DM|ML*	*DM|DB*	*DM|ML|DB*	*Conf|Journal*	Average
WCN	1.1628±0.1747	1.3423±0.3142	1.3191±0.2785	1.3371±0.3208	1.2904
WAA	1.1658±0.1346	1.2933±0.1779	1.2777±0.1766	1.3050±0.2653	1.2605
HPLP	1.0557±0.1507	1.2127±0.1894	1.3231±0.1894	1.2010±0.3339	1.1981
NMF	1.4363±0.1698	1.4960±0.1948	1.5044±0.1954	1.5192±0.3954	1.4890
CP-Tensor	2.0409±0.3687	2.4127±0.4796	2.2487±0.4194	2.6458±0.5991	2.3370
CoEvol	**0.7400**±0.0991	**0.9135**±0.1512	**0.9650**±0.1849	**0.7754**±0.1459	0.8485

5.2.3 Finding Cross-Network Lag Correlations

The formation and evolution of network structures in one network are often closely correlated with that in another. Specifically, one can view $\boldsymbol{V}_1(t) \ldots \boldsymbol{V}_s(t)$ as a set of s different groups of $n \times k$ co-evolving time-series. A one-to-one correspondence exists between the $(p, q)^{th}$ entry of the matrix $\boldsymbol{V}_a(t)$ and the $(p, q)^{th}$ entry of the matrix $\boldsymbol{V}_b(t)$ for $a \neq b$. Therefore, a natural question arises as whether one can find groups of nodes such that evolution in one network is highly predictive of evolution in another. We informally state this problem as the lag-centric evolution prediction problem. In the following, we discuss a simple algorithm to discover lag correlations between different networks.

1. For each node-latent-component (i, q), where $i \in [1, n]$ and $q \in [1, k]$, and for each network pair $(a, b) \in [1, s]$, we compute the Pearson correlation coefficient between the $(i, q)^{th}$ component of $V_a(t + \delta)$ and $(i, q)^{th}$ component of $V_b(t)$ over various values of t. We store all values of (a, b) and (i, q) such that the correlation is larger than a given threshold ρ.

2. For each network pair (a, b), we report the subset of components $C_\rho(a, b)$ which satisfy the condition of the first step. This is a closely related group of components which time-series in the network a are highly related to the other in network b.

The calculation of these lag correlations is time-efficient. Assume that we have s networks, the latent dimension is k, lag time-stamp is set to $[-\delta, \delta]$. Under this setting, it's obvious that

we have $n \times k$ components. For each component, one has to slide $2 \times \delta$ times between any two networks. We have $s \times (s-1)$ pairs of networks in total. Thus the complexity of the calculation is $2\delta nks(s-1)$. Usually, we set δ and k to a relative small number. Thus the time complexity is $O(s^2 n)$. If a dataset contains a small number of networks, then the complexity goes to $O(n)$.

5.2.3.1 Re-visit Cross-Network Link Prediction

One application that utilize the detected lag correlations is to help predict the cross-network structure. In this section, we detect lag correlation set $C_\rho(a, b)$ on dataset *DM|DB* to help improve the prediction RMSE at time-stamp T.

We build the time-series with the factors $\boldsymbol{V}_r(t)$ for each network r. Factors $\boldsymbol{V}_r(t)$ at different time-stamps t ($t \in [1, T-1]$) are computed by our model CoEvol. Thus for each network r, we obtain $n \times k$ times-series with length $T-1$. Each time-series at node-latent-component pair (i, q) (denoted as $\mathcal{S}_{i,q}$) is a $T-1$ vector which has elements $V_{i,q}^{(r)}(1), ..., V_{i,q}^{(r)}(T-1)$.

For each of the $n \times k$ times-series in *DM* network, we calculate the correlation with its counterpart in *DB* network. The maximum correlation among all lag-years ($[-15, -1] \cup [1, 15]$) is saved for each component. Fig. 5.9 shows the distributions of lag correlations and lag years. It shows that most of node-latent-component pairs have a positive correlation between *DM* and *DB* networks. This is reasonable since the authors may share similar co-authorship patterns in data mining network and database network. The lag year distribution is "symmetric" between negative lag years and positive lag years. It demonstrates that the data mining network is highly predictive of evolution in the database network and vice versa. Additionally, the frequency distribution follows a convex curve in the lag year range $[-15, 0]$ and $[0, 15]$, respectively. These curves depict that as lag year increases, the correlation decreases ($\delta \in [1, 10]$). However, if the lag year is too large, the length of time-series $\mathcal{S}_{i,q}$ in *DM* and *DB* networks is very short which will cause overfitting. Therefore the correlation increases when $|\delta|$ is large.

Cross-network link prediction is possible only if lag correlations exist between pairs of networks for $\delta > 0$. Therefore, the first step is to identify such pairs of networks by using the approach discussed in the Section 5.2.3. Now, we will show that this lag correlation information

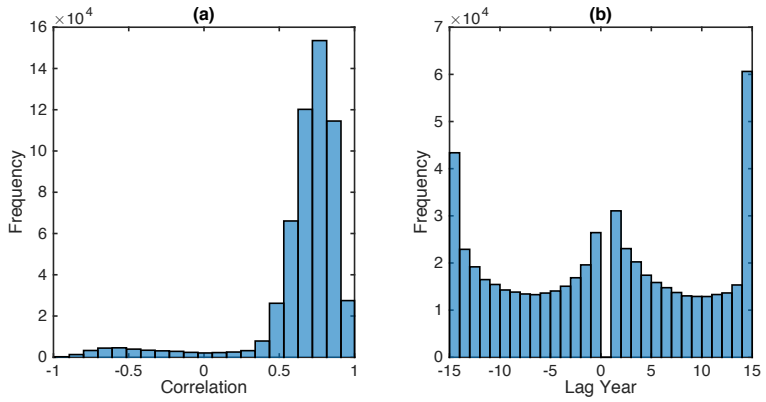


Figure 5.9: Lag correlation distribution and lag year distribution of dataset DM|DB

can improve the link prediction results of CoEvol framework. The task is to predict the last time-stamp ($t = T$) network structure of dataset *DM|DB*. In Section 5.2.2.2, the last time-stamp network structure of network $\boldsymbol{A}_r(T)$ is predicted by calculating $\boldsymbol{U}\boldsymbol{V}_r(T)^T$, where $\boldsymbol{U}$ and $\boldsymbol{V}_r(T)$ are learned by the data from first $T-1$ time-stamps. But now we are trying to enhance the prediction of $\boldsymbol{V}_r(T)$ with lag correlation information. The experiment is designed as follows: 1) find the highly correlated node-latent-component pairs, and save their indices (i, q). Since the dataset *DM|DB* only has two networks, there is no need to save the network pair information. 2) if any node-latent-component pair (i, q) has a lag correlation larger than a given threshold ρ, we use a linear regression model to fit the time-series indexed by (i, q), and predict the last time-stamp value of the lagged network. This prediction value is set as the corresponding entry in $(\boldsymbol{V}_r(T))_{i,q}$ (r is determined based on positive lag or negative lag). We then calculate the RMSE based on the modified $\boldsymbol{V}_r(T)$ and the constant matrix $\boldsymbol{U}$.

The parameters are set as follows: lag year $\delta = 3$, lag correlation $\rho \in [0.85, 1]$ with interval 0.01. Fig. 5.10 shows the accuracy of link weight prediction on dataset *DM|DB*, the x-axis is the correlations, the y-axis shows the RMSE (left) and the number of node-latent-components (right). The blue horizontal line is the prediction accuracy of CoEvol without incorporating the lag correlation information. As can be seen in this figure, when $\delta > 0.95$, the prediction RMSE is better than the RMSE obtained by the vanilla CoEvol. It shows that when the latent components have a very high correlation (> 0.95), we can just use this correlation information to predict it's behavior in the near future. However, if the correlation is low (< 0.95 in this experiment setting),

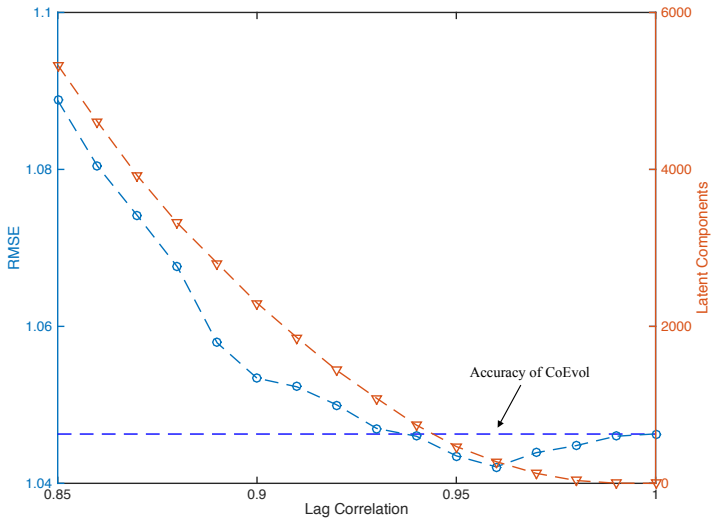


Figure 5.10: Prediction RMSE and latent components under different lag correlation thresholds

the RMSE increases. The reason is that, as more and more latent components meet the threshold of ρ, it introduces more noise into the estimation matrix $\boldsymbol{V}_r(T)$.

5.2.4 Cross-Network Community Detection

Information networks evolve over time, driven by the shared properties of each individual network, such as the similarity of node attributes and the arrangement of links. Therefore, the proposed framework CoEvol can also be used to detect communities across networks by exploring the correlations among multiple networks. Actually such correlation information can be learned from the factor $\boldsymbol{U}$ from Eq. (5.29). A typical approach to detect such groups (communities) is to propose application-related objective functions [Fortunato, 2010] to minimize the internal connectivity while maximizing the external connectivity. As to the CoEvol framework, it naturally has clear clustering effects [**?**]. Thus we can use a cluster-based method to detect communities on the learned factor $\boldsymbol{U}$, which capitalizes on the large number of potential interactions between nodes. The simple method used to discover communities between different networks is summarized as follows,

1. Compute factor matrix $\boldsymbol{U}$ by Algorithm 5 with input networks $\{\boldsymbol{A}_r(t)\}_{r=1}^{s}$.

2. Apply clustering algorithms on matrix $\boldsymbol{U}$. We treat $\boldsymbol{U}$ as a dataset with n samples, each of

which has k dimensions. The detected clusters are resultant communities.

5.2.4.1 Compared Methods and Evaluation

In this section, we use three canonical community detection methods as baselines: Information Maps (Infomap) [Rosvall and Bergstrom, 2007], Multilevel [Blondel et al., 2008] and Leading Eigenvector (Eigen) [Newman, 2006a]. We evaluate the community detection results by a simple and transparent measure *purity*. To compute purity, each community is assigned to the label which is most frequent in the cluster, and then the accuracy of this assignment is measured by counting the number of correctly assigned nodes and dividing by n. Formally, $purity(\mathcal{C}, \mathcal{L}) = \frac{1}{n}\sum_i \max_j |\mathcal{C}_i \cap \mathcal{L}_j|$, where $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_I\}$ is the set of communities, and $\mathcal{L} = \{\mathcal{L}_1, \mathcal{L}, ..., \mathcal{L}_J\}$ is the set of labels such as KDD, VLDB and ICML. We interpret $\mathcal{C}_i$, $\mathcal{L}_j$ as the set of nodes.

5.2.4.2 Experimental Results

In this section, we look at the community detection results on both synthetic and real-world datasets.

Community Detection on Synthetic Dataset: We create two large connected components with 2242 and 3262 nodes respectively. The edges are randomly assigned in two networks $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$. Here we only generate data with one time-stamp for the ease of result visualization. The rank of factorization k is set to 2. Other parameter settings remain same as in Section 5.2.2.2. The left subfigure in Fig. 5.11 visualizes the adjacency matrix $\boldsymbol{A}$, where $\boldsymbol{A} = \boldsymbol{A}_1 + \boldsymbol{A}_2$, and the right one shows the corresponding shared factor $\boldsymbol{U}$ learned by CoEvol. In our experiment, the multidimensional data set is clustered corresponding to the rows of the shared factor matrix $\boldsymbol{U}$. But the learned matrix $\boldsymbol{U}$ in Fig. 5.11 doesn't show a good property to be clustered (different colors represent different components each edge comes from). However, when we factorize $\boldsymbol{A}\boldsymbol{A}^T$ instead of $\boldsymbol{A}$, all the points in the learned matrix $\boldsymbol{U}$ have been pulled to the two axes which correspond to two clusters, as shown in Fig. 5.12.

Fig. 5.12 depicts that the shared factor $\boldsymbol{U}$ is easily to be clustered into two groups. The reason is that $\boldsymbol{A}\boldsymbol{A}^T$ is a more robust similarity measure between a pair of nodes. Each entry in $\boldsymbol{A}\boldsymbol{A}^T$

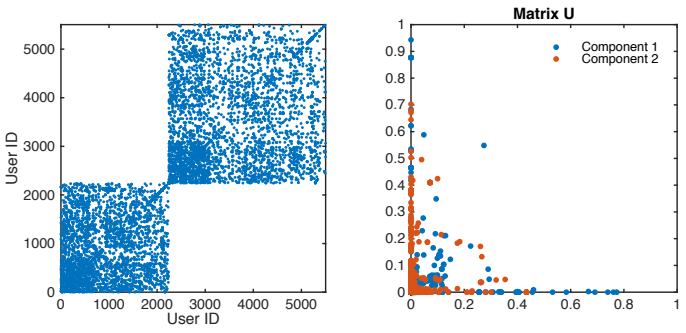


Figure 5.11: Visualization of synthetic data with two components and the corresponding factor $\boldsymbol{U}$

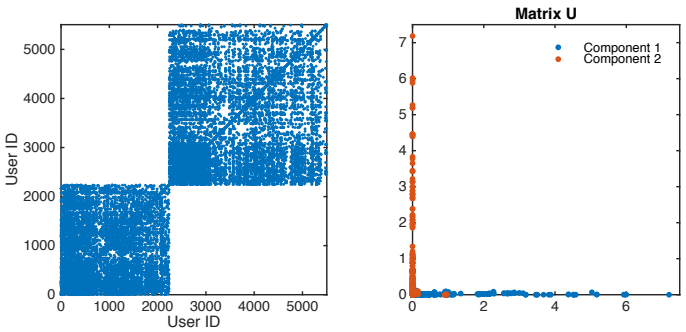


Figure 5.12: Visualization of matrix $\boldsymbol{A}\boldsymbol{A}^T$ and the corresponding factor $\boldsymbol{U}$

represents the number of common neighbors between a pair of nodes. If $\boldsymbol{A}$ is originally represented as a binary matrix, the number of common neighbors between a pair of nodes shows their clustering characteristics [Liben-Nowell and Kleinberg, 2007]. In summary, using $\boldsymbol{A}\boldsymbol{A}^T$ is a way to augment the matrix $\boldsymbol{A}$ with neighborhood information, thus it is more robust. This is different from cross-network link prediction in Section 5.2.2 which is more natural to use the original matrix $\boldsymbol{A}$. Because link prediction tries to predict the new links or link weights rather than similarities. In the following subsection, we will factorize $\boldsymbol{A}\boldsymbol{A}^T$ instead of $\boldsymbol{A}$.

Table 5.7: Purity of cross-network community detection

Methods	*DM*|*ML*	*DM*|*DB*	*DM*|*ML*|*DB*	*Conf*|*Journal*
Infomap	0.7716	0.7801	0.7431	0.8063
Multilevel	0.7563	0.7650	0.7218	0.7836
Eigen	0.7283	0.6909	0.7060	0.7621
CoEvol	**0.7904**	**0.7814**	**0.7559**	**0.8148**

Community Detection on Real-world Datasets: In this section, we report the cluster purity of the detected communities for each methods. Each node (author) in these four datasets is labeled by the venues (e.g. KDD, VLDB etc.) where the author published the majority papers. The cluster purity score tends to be higher if we set more clusters for the k-means algorithm. In order to have a fair comparison, the cluster numbers are set to $50, 50, 100, 2500$ for datasets *DM|ML*, *DM|DB*, *DM|ML|DB* and *Conf|Journal*, respectively. The other parameters remains the same as in Section 5.2.2.2. Table 5.7 shows the cluster purity of each method and the number of detected communities at the last time-stamp. It can be seen that our model obtains the highest purity on all datasets. Notably, Infomap detects more communities, each with a small number of community members, which tends to achieve a high purity according to the definition. But our model still preforms better than Infomap.

5.3 Summary

In this chapter, we introduce two novel time-dependent matrix factorization based models, TMF and COEVOL, for evolutionary network analysis. These model have the advantage of significant generality in addressing various temporal applications because of its ability to explicitly represent the networks as a function of time. As specific examples, we provide results for temporal weight trend prediction, link prediction, dynamic community detection and event detection within the TMF framework, and we evaluate the extraordinary generality of COEVOL in terms of its applicability to cross-network link prediction, lag correlation detection and community detection tasks. Even though we provide more general models, our results show that their specific *instantiations* to different prediction tasks perform better than state-of-the-art techniques, thereby demonstrates the generality and effectiveness of the TMF and COEVOL frameworks.

CHAPTER 6

Evolutionary Network Analysis: Applications

Massive and dynamic networks arise in many practical areas such as social media, security and public health. In this chapter, we will talk about two applications, namely anomaly detection and link prediction, under the dynamic setting. Given an evolutionary network, it is crucial to detect structural anomalies, such as vertices and edges whose "behaviors" deviate from underlying majority of the network, in a real-time fashion. Section 6.1 proposes a novel approach, NETWALK, for anomaly detection in dynamic networks by learning network representations which can be updated dynamically as the network evolves. The vertices of the dynamic network are encoded to vector representations by *clique embedding*, which jointly minimizes the pairwise distance of vertex representations of each walk derived from the dynamic networks, and the deep autoencoder reconstruction error serving as a global regularization. The vector representations can be computed with constant space requirements using *reservoir sampling*. On the basis of the learned low-dimensional vertex representations, a clustering-based technique is employed to incrementally and dynamically detect network anomalies.

Link prediction in dynamic networks has also attracted tremendous research interests. Many models have been developed to predict links that may emerge in the immediate future from the past evolution of the networks. There are two key factors: 1) a node is more likely to form a link in the near future with another node within its close proximity, rather than with a random node; 2) a dynamic network usually evolves smoothly. Existing approaches seldom unify these two factors to strive for the spatial and temporal consistency in a dynamic network. To address this limitation, Section 6.2 introduces the LIST model to predict links in a sequence of networks over time. LIST characterizes the network dynamics as a function of time, which integrates the spatial topology of network at each timestamp and the temporal network evolution.

6.1 Anomaly Detection with Dynamic Network Embedding

Anomaly detection (a.k.a outlier detection) in dynamically changing networks is a long-standing problem deeply motivated in a number of application domains, such as social media, security, public health, and computational biology [Aggarwal, 2013, Akoglu et al., 2015, Gupta et al., 2014a, Ranshous et al., 2015, Akoglu and Faloutsos, 2013]. Identifying time-varying anomalies (such as edges or vertices), which represent significant deviations from "normal" structural patterns in the evolving network, can shed important light on the functional status of the whole system. Many methods have been proposed in the past decade to solve this problem [Aggarwal et al., 2011, Gupta et al., 2012b, Manzoor et al., 2016, Ranshous et al., 2016, Yu et al., 2017b]. Some prominent examples of applications are summarized as follows.

- With the popularity of social media, anomalous behaviors can be found in the underlying social network. The malicious activities such as cyber-bullying, terrorist attack planning and fraud information dissemination can be detected as anomalies using anomaly detection models based on social network.

- The advanced persistent threat (APT) detection problem in security can also be cast as real-time anomaly detection in network streams. In an APT scenario, we are given a stream of system logs which can be used to construct information-flow networks. Information flows induced by malicious activities can be quite different from normal system behaviors.

- In clinics, anomaly detection can provide valuable information on managing and diagnosis with the electric patient records. The data typically consist of records from various types of entities (vertices) such as patients, symptoms and treatments, which can be modeled as a multi-partite network representing their complex interactions. Anomalies in such networks can pinpoint important scenarios requiring instant human interventions, such as abnormal patient condition or recording errors.

To detect network anomalies in these applications, a typical approach is to first perform network sketching and then identify anomalies in the sketches through clustering and outlier detection, such as in [Manzoor et al., 2016, Ranshous et al., 2016]. The network sketches serve

as a compact, latent representation of the network and thus allow efficient updates as new network objects arrive in a streaming fashion, without having to maintain the complete details of the full network. In the literature, the network sketches have been learned through locality-sensitive hashing [Indyk and Motwani, 1998] and count-min sketch [Cormode and Muthukrishnan, 2005]. However, these approaches are not directly designed to learn the network sketches that can simultaneously preserve important structural relations, such as the local neighborhood composition or proximity landscapes. Thus, the sketches extracted are usually shallow [Bengio et al., 2003, Levy and Goldberg, 2014], and thereby bottleneck the accuracy of downstream tasks such as anomaly detection.

Recently, network embedding has attracted significant interest and shown promising results, in particular towards obtaining desired low-dimensional network representations that better preserve the neighborhood information [Bengio et al., 2003, Levy and Goldberg, 2014, Mikolov et al., 2013a, Mikolov et al., 2013b]. The structure preserving property of the network embedding makes it particularly suitable for anomaly detection tasks, by examining the similarity between vertices/edges in the latent representation. For example, vertices staying far away from the majority clusters in the multidimensional latent space will very likely indicate certain types of anomalies, which can be detected conveniently through dynamic clustering algorithms. However, existing methods for network embedding can not update the representation dynamically as new vertices or edges keep feeding, and thus may not be perfectly suitable for anomaly detection in a dynamic environment [Bengio et al., 2003, Levy and Goldberg, 2014, Mikolov et al., 2013a, Mikolov et al., 2013b]. In case of a rapidly evolving network, the problem can be even more challenging. It is therefore highly desirable to design an effective and especially efficient embedding algorithm that is capable of fast, real-time detection with bounded memory usage.

To address this problem, in this study, we propose the NETWALK algorithm to incrementally learn network representations as the network evolves, and detect anomalies in the networks in a real-time fashion. Figure 6.1 shows an illustrative diagram of the anomaly detection pipeline in dynamic networks. First, we learn the latent network representation by using a number of network walks extracted from the initial network. The representation is obtained not only through maintaining the pairwise vertex-distance in the local walks, but also by hybridizing it with the hidden layer

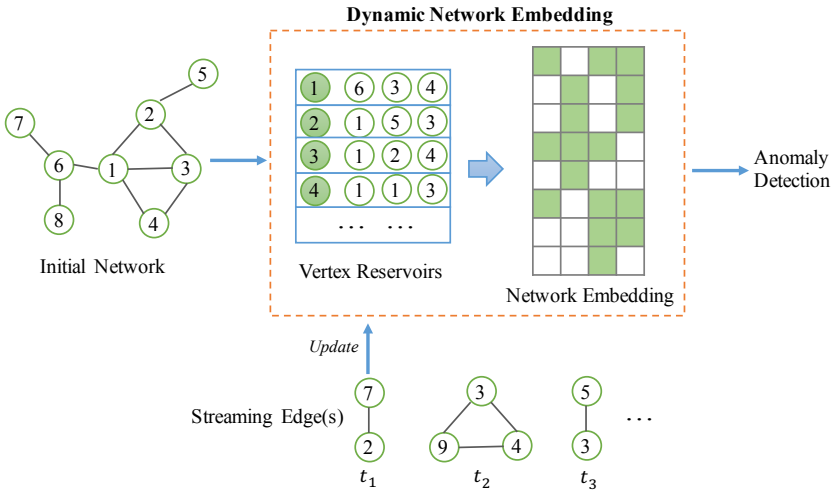


Figure 6.1: Workflow of NETWALK for anomaly detection in dynamic networks

of a deep autoencoder, such that the resultant embedding is guaranteed to faithfully reconstruct the original network. By doing this, the learned vertex coordinates in the multi-dimensional Euclidean space can achieve both local fitting and global regularization. In addition, the learned representations can be easily updated over dynamic changes by leveraging a *reservoir sampling* strategy. Then, a dynamic clustering model is used to flag anomalous vertices or edges based on the learned vertex or edge representations. We quantitatively validate the effectiveness and efficiency of the proposed framework on real datasets.

6.1.1 Problem Formulation

Given a temporal network $\mathcal{G}(t) = (\mathcal{E}(t), \mathcal{V}(t))$, we assume that the incoming stream of network objects at time-stamp t is typically a small number of network objects denoted by an edge set[1] $E^{(t)}$ where $|E^{(t)}| \geq 1$. All vertices in the edge set $E^{(t)}$ at time-stamp t are denoted by $V^{(t)}$. The vertex set $\mathcal{V}(t)$ denotes the union of the vertex sets across all time-stamps from 1 to t, that is, $\mathcal{V}(t) = \cup\{V^{(i)}\}_{i=1}^{t}$. Similarly, we have $\mathcal{E}(t) = \cup\{E^{(i)}\}_{i=1}^{t}$. Note that the complete set of vertices may not be known at time-stamp t, since new vertices may keep arriving at time-stamp t' for any $t' > t$. The network $\mathcal{G}(t)$ includes all edges received from time-stamps 1 to t.

[1]Note that the case of incoming stream of edges includes the case of new vertices since an edge contains both the edge itself and the connected vertices. The case of incoming stream of singleton vertices is trivial because they are obviously anomalies.

Our goal is to detect anomalous vertices, edges and communities (group of vertices) at any given time-stamp t, i.e., in real time as $E^{(t)}$ occurs. To achieve this goal, we encode the network $\mathcal{G}(t)$ as a feature matrix, where each row is the vector representation of a vertex (Section 6.1.2). The main challenges are, i) we need a cohesive way to encode the dynamic network, ii) incoming network objects should be easily coded with the learned network representations, iii) network representations need to be efficiently updated as new network objects arrive. We then follow a clustering-based approach to detect the anomalies in the dynamic network (Section 6.1.3). The clusters are generated after calculating the distances between the embedded vertices. Whether the incoming network objects are anomalies or not can be determined from the distance between their respective representations and existing clusters. The clustering results are updated efficiently as new network objects arrive.

6.1.2 Encoding network Streams

In order to detect anomalies in dynamic networks in real time, our method needs to learn network representations and perform online updates efficiently as network evolves. For clarity, now we only discuss the case that new edges stream in on unweighed network. The cases of decreasing edges or tackling weighted networks are similar and will be discussed in Section 6.1.2.2. In this section, we present the network encoding and the updating phase in NETWALK which does not require to store the entire network.

6.1.2.1 Walk Generation

Analogous to word embedding techniques [Mikolov et al., 2013a, Mikolov et al., 2013b] in constructing vector representations, we decompose the network into a set of network walks each of which contains a list of vertices selected by a random walk[2]. We formally define the network walk as follows.

Definition 3 (Network Walk)**.** *For a given vertex $v_1 \in \mathcal{V}$ in a network $\mathcal{G}(\mathcal{E}, \mathcal{V})$, its network walk*

[2]Note that this works for both directed and undirected network

set is defined as $\Omega_{v_1} = \{(v_1, v_2, ..., v_l) |\ (v_i, v_{i+1}) \in \mathcal{E} \wedge p(v_i, v_{i+1}) = \frac{1}{D_{v_i,v_i}}\}$*, which is a collection of* l*-hop walks starting from vertex* v_1*. The transition probability* $p(v_i, v_{i+1})$ *from* v_i *to* v_{i+1} *is proportional to the degree* D_{v_i,v_i} *of vertex* v_i*. We call* Ω_v *a network walk set starting from* v*, and* $\Omega = \{\Omega_v\}_{v \in \mathcal{V}}$ *as the union of all walks.*

Similar to the word frequency which typically follows a power law distribution in natural language, we observe that if the degree distribution of the network follows a power law distribution, the frequency distribution of vertices occurring in the network walks also follows a power law distribution (or Zipf's law) [Perozzi et al., 2014b]. Therefore we will use a stream of network walks as our basic tool for extracting information from a network. We then learn the vertex representations of the network using a novel embedding method introduced in Section 6.1.2.2.

6.1.2.2 Learning Network Representations

We formulate the network representation learning problem as an optimization problem. Our goal is to learn a mapping function $f : \mathcal{V} \rightarrow \mathbb{R}^d$ such that each $v \in \mathcal{V}$ is represented as a d-dimensional vector, where d is the latent-space dimension. The mapping function f applies to any (un)directed, (un)weighted network.

Inspired by skip-gram architecture [Mikolov et al., 2013b], we propose a network embedding algorithm, *clique embedding*, that utilizes an deep auto-encoder neural network to learn the vector representation of vertices through a stream of network walks while minimizing the pairwise distance among all vertices in each walk. Figure 6.2 depicts the clique embedding model used in NETWALK. The inputs and outputs are one-hot encoded vectors, that is, for a given vertex input $\mathbf{x}_p^{(i)} \in \mathbb{R}^n$, only one out of n elements will be 1, and all others are 0's. Our goal is to learn a latent representation for each input network walk $\{\mathbf{x}_p^{(i)}\}_{p=1}^{l}$. Here l is the walk length, $\{\mathbf{W}^{(\ell)}\}_{\ell=1}^{n_l}$ are the weight matrices, $\{\mathbf{b}^{(\ell)}\}_{\ell=1}^{n_l}$ are the bias vectors, and $f^{(\ell)}(\cdot)$ denotes the output of each layer.

Formally, given a one-hot encoded network walk $\{\mathbf{x}_p^{(i)}\}_{p=1}^{l}$, $i = 1, ..., |\Omega|$, we want to learn the following representations in a n_l-layer autoencoder network,

$$f^{(\frac{n_l}{2})}(\mathbf{x}_p^{(i)}) = \sigma(\mathbf{W}^{(\frac{n_l}{2})\top} h^{(\frac{n_l}{2})}(\mathbf{x}_p^{(i)}) + \mathbf{b}^{(\frac{n_l}{2})}), \tag{6.1}$$

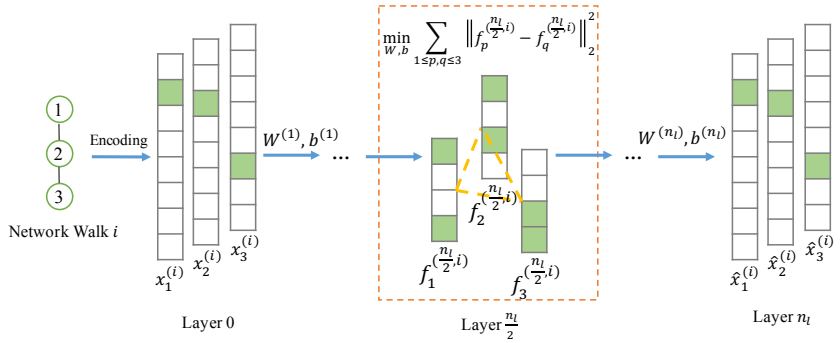


Figure 6.2: Illustration of clique embedding for one network walk of length three

where

$$h^{(\frac{n_l}{2})}(\mathbf{x}_p^{(i)}) = \mathbf{W}^{(\frac{n_l}{2}-1)} f^{(\frac{n_l}{2}-1)}(\mathbf{x}_p^{(i)}) + \mathbf{b}^{(\frac{n_l}{2}-1)}. \tag{6.2}$$

Here, $\sigma(z) = \frac{1}{1+\exp(z)}$ is the sigmoid function; $n_l \geq 2$; $f^{(0)}(\mathbf{x}_p^{(i)}) = \mathbf{x}_p^{(i)}$. In an auto-encoder network, the output hypotheses $f^{(n_l)}(\mathbf{x}_p^{(i)})$ is approximately equal to $\mathbf{x}_p^{(i)}$. Therefore, if we use ℓ_2 norm to minimize the reconstruction error, the objective function becomes

$$J_{AE} = \frac{1}{2} \sum_{i=1}^{|\Omega|} \sum_{p=1}^{l} \left\| f^{(n_l)}(\mathbf{x}_p^{(i)}) - \mathbf{x}_p^{(i)} \right\|_2^2. \tag{6.3}$$

We also seek to minimize the pairwise distance among all vertices[3] of each network walk in the embedding space at layer $\frac{n_l}{2}$, which can be formally described as follows,

$$J_{Clique} = \sum_{i=1}^{|\Omega|} \sum_{1 \le p,q \le l} \left\| f^{(\frac{n_l}{2})}(\mathbf{x}_p^{(i)}) - f^{(\frac{n_l}{2})}(\mathbf{x}_q^{(i)}) \right\|_2^2. \tag{6.4}$$

Due to the sparsity of the input and output vectors, we consider a sparse auto-encoder with sparsity parameter ρ and penalize it with the Kullback-Leibler divergence [Ng, 2011],

$$\mathrm{KL}(\rho \| \hat{\boldsymbol{\rho}}^{(\ell)}) = \sum_{j=1}^{d} \mathrm{KL}(\rho \| \hat{\rho}_j^{(\ell)}) = \sum_{j=1}^{d} \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j}, \tag{6.5}$$

where $\hat{\boldsymbol{\rho}}^{(\ell)} = \frac{1}{|\Omega| \times l} \sum_{i=1}^{|\Omega|} \sum_{p=1}^{l} f^{(\ell)}(\mathbf{x}_p^{(i)})$ is the average activation of the units in the hidden layer. This sparsity constraint penalizes large deviation of $\hat{\rho}_j^{(\ell)}$ from ρ. Given a training set of network

[3]This is inspired by skip-gram architecture that considers all word pairs within a distance window. It is effective for extracting local proximity information [Mikolov et al., 2013b].

walks Ω, we then define the overall cost function to be:

$$J(\mathbf{W},\mathbf{b}) = \underbrace{\sum_{i=1}^{|\Omega|}\sum_{1\le p,q\le l}\left\|f^{(\frac{n_l}{2})}(\mathbf{x}_p^{(i)}) - f^{(\frac{n_l}{2})}(\mathbf{x}_q^{(i)})\right\|_2^2}_{\text{Clique Embedding Loss}} + \underbrace{\frac{\gamma}{2}\sum_{i=1}^{|\Omega|}\sum_{p=1}^{l}\left\|f^{(n_l)}(\mathbf{x}_p^{(i)}) - \mathbf{x}_p^{(i)}\right\|_2^2}_{\text{Reconstruction Error}}$$
$$+ \underbrace{\beta\sum_{\ell=1}^{n_l-1}\sum_{j}\mathrm{KL}\left(\rho\|\hat{\rho}_j^{(\ell)}\right)}_{\text{Sparsity Constraint}} + \underbrace{\frac{\lambda}{2}\sum_{\ell=1}^{n_l}\left\|\mathbf{W}^{(\ell)}\right\|_F^2}_{\text{Weight Decay}}, \tag{6.6}$$

where $|\Omega|$ is the number of network walks, l is the walk length. The weight decay term decreases the magnitude of the weights, and helps prevent overfitting. γ, β and λ control the weight of the corresponding penalty terms. The loss function $J(\mathbf{W},\mathbf{b})$ can also be written in a matrix form,

$$J(\mathbf{W},\mathbf{b}) = \sum_{i=1}^{|\Omega|}\mathrm{Tr}(\boldsymbol{\mathcal{F}}^{(i)}\mathbf{L}\boldsymbol{\mathcal{F}}^{(i)\top}) + \frac{\gamma}{2}\left\|\boldsymbol{\mathcal{H}}^{(n_l)}(\mathbf{X}) - \mathbf{X}\right\|_F^2 + \beta\sum_{\ell=1}^{n_l-1}\mathrm{KL}(\rho\|\hat{\boldsymbol{\rho}}^{(\ell)}) + \frac{\lambda}{2}\sum_{\ell=1}^{n_l}\left\|\mathbf{W}^{(\ell)}\right\|_F^2, \tag{6.7}$$

where $\boldsymbol{\mathcal{F}}^{(i)} = [f_1^{(i)}, f_2^{(i)}, ..., f_l^{(i)}]$, $f_l^{(i)} = f^{(\frac{n_l}{2})}(\mathbf{x}_l^{(i)})$; $\mathbf{L}$ is the Laplacian matrix of the clique with l vertices, thus we have $\mathbf{L} = \mathbf{I}_l \times (l-1) - \mathbf{\Phi}$, and $\mathbf{\Phi}_{i,j} = 1, \forall i \neq j$. $\mathbf{X} = [\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(|\Omega|)}]$, $\mathbf{x}^{(i)} = [\mathbf{x}_1^{(i)}, \mathbf{x}_2^{(i)}, ..., \mathbf{x}_l^{(i)}]$; $\boldsymbol{\mathcal{H}}^{(n_l)}(\mathbf{X}) = [g^{(1)}, g^{(2)}, ..., g^{(|\Omega|)}]$, $g^{(i)} = [f^{(n_l)}(\mathbf{x}_1^{(i)}), f^{(n_l)}(\mathbf{x}_2^{(i)}), ...f^{(n_l)}(\mathbf{x}_l^{(i)})]$.

Our goal is to minimize $J(\mathbf{W},\mathbf{b})$ as a function of $\mathbf{W}$ and $\mathbf{b}$. The key step is to compute the partial derivatives of objective function Eq. (6.7), with respect to $\mathbf{W}$ and $\mathbf{b}$ to derive updates. Inspired by back-propagation algorithm [**?**], we introduce "error terms" for $\delta^{(\ell)}$ and $\delta^{(n_l)}$ for the hidden layer and output layer respectively. These "error terms" can be computed as follows:

$$\begin{aligned}
\delta^{(n_l)} &= -\gamma\nabla_{f^{(n_l)}(\mathbf{X})}J_{AE} \circ \sigma'\left(h^{(n_l)}(\mathbf{X})\right) \\
&= -\gamma\left(\boldsymbol{\mathcal{H}}^{(n_l)}(\mathbf{X}) - \mathbf{X}\right) \circ f^{(n_l)}(\mathbf{X}) \circ \left(1 - f^{(n_l)}(\mathbf{X})\right), \qquad (6.8)\\
\delta^{(\ell)} &= \left(\mathbf{W}^{(\ell)\top}\delta^{(\ell+1)} + \beta\frac{\hat{\boldsymbol{\rho}}^{(\ell)} - \boldsymbol{\rho}_0}{\hat{\boldsymbol{\rho}}^{(\ell)}(1-\boldsymbol{\rho}_0)}\right) \circ \sigma'\left(h^{(\ell)}(\mathbf{X})\right) \\
&= \left(\mathbf{W}^{(\ell)\top}\delta^{(\ell+1)} + \beta\frac{\hat{\boldsymbol{\rho}}^{(\ell)} - \boldsymbol{\rho}_0}{\hat{\boldsymbol{\rho}}^{(\ell)}(1-\boldsymbol{\rho}_0)}\right) \circ f^{(\ell)}(\mathbf{X}) \circ \left(1 - f^{(\ell)}(\mathbf{X})\right), \qquad (6.9)
\end{aligned}$$

where $\boldsymbol{\rho}_0$ is a vector with all entries ρ; "$\circ$" denotes the element-wise product. Since the *clique*

embedding loss only depends on $\{\mathbf{W}^{(\ell)}, \mathbf{b}^{(\ell)}\}_{\ell=1}^{\frac{n_l}{2}}$, then the derivatives for $\ell > \frac{n_l}{2}$ are

$$\nabla_{\mathbf{W}^{(\ell)}} J(\mathbf{W}, \mathbf{b}) = \delta^{(\ell)} \left(f^{(\ell-1)}(\mathbf{X})\right)^\top + \lambda \mathbf{W}^{(\ell)}, \tag{6.10}$$

$$\nabla_{\mathbf{b}^{(\ell)}} J(\mathbf{W}, \mathbf{b}) = \sum_{i=1}^{|\Omega|} \delta_i^{(\ell)}. \tag{6.11}$$

We need to take the clique embedding loss into consideration when computing the derivatives for $\{\mathbf{W}^{(\ell)}, \mathbf{b}^{(\ell)}\}_{\ell=1}^{\frac{n_l}{2}}$.

$$\begin{aligned}\nabla_{\mathbf{W}^{(\ell)}} J(\mathbf{W}, \mathbf{b}) = &\sum_{i=1}^{|\Omega|} \boldsymbol{\mathcal{F}}^{(i)} (\mathbf{L} + \mathbf{L}^\top) \circ \boldsymbol{\mathcal{F}}^{(i)} \circ (1 - \boldsymbol{\mathcal{F}}^{(i)}) \left(f^{(\ell-1)}(\mathbf{X})\right)^\top \\ &+ \delta^{(\ell)} \left(f^{(\ell-1)}(\mathbf{X})\right)^\top + \lambda \mathbf{W}^{(\ell)},\end{aligned} \tag{6.12}$$

$$\nabla_{\mathbf{b}^{(\ell)}} J(\mathbf{W}, \mathbf{b}) = \sum_{i=1}^{|\Omega|} \boldsymbol{\mathcal{F}}^{(i)} (\mathbf{L} + \mathbf{L}^\top) \circ \boldsymbol{\mathcal{F}}^{(i)} \circ (1 - \boldsymbol{\mathcal{F}}^{(i)}) + \delta_i^{(\ell)}. \tag{6.13}$$

Starting from every vertex $v \in V$, we generate all network walks via random walk. Then network representations are learned by optimizing the aforementioned loss function $J(\mathbf{W}, \mathbf{b})$. The pseudocode for network encoding is given in Algorithm 6.

In a fully streaming setting, the entire vertex set $\mathcal{V}$ will change over time, and hence is the number of vertices n. In this case, we need to pre-allocate a fixed length for one-hot encoding technique to encode the input vectors $\mathbf{x}_p^{(i)}$'s.

Visualization. In the network representation learning phase, NETWALK takes an initial network as input and learns a latent representation for every vertex. Here, we show the encoding capability of clique embedding in NETWALK by applying our method to the Zachary's karate network [Zachary, 1977]. Figure 6.3(a) shows the original network in which the vertex color indicates the community to which each vertex belongs. Figure 6.3(b) presents the 2-dimensional representations learned by NETWALK. Notably, the linearly separable clusters can be found in the vector representation space learned by our method.

6.1.2.3 Edge Encoding

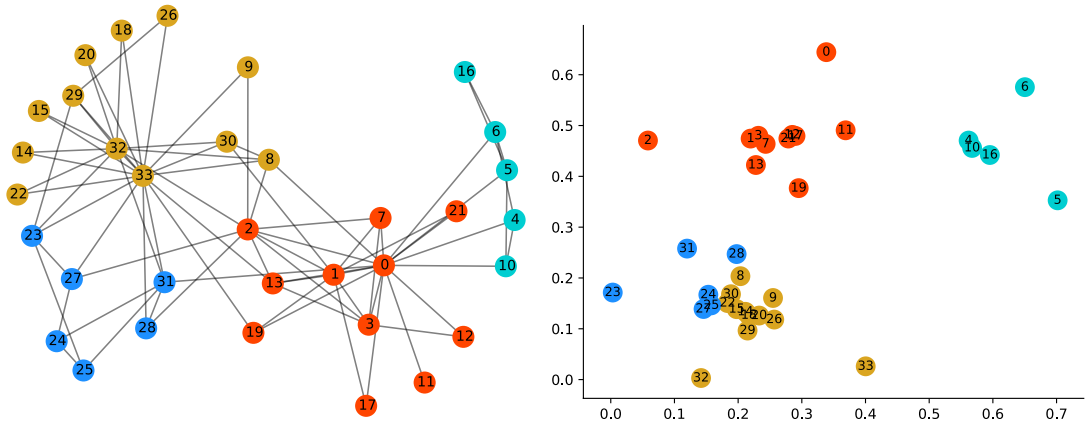
NETWALK learns vector representations for vertices, which allows us to detect vertex anomalies based on clustering. In addition, we are also interested in edge anomaly detection. Therefore, in

Algorithm 6 Clique Embedding of NETWALK

Input: Network walk set Ω.

Output: Network representations $f^{\frac{n_l}{2}}(\mathbf{x}_p^{(i)})$

Set latent dimension d, sparsity ρ, weight control parameters γ, β and λ.
Randomly initialize $\{\mathbf{W}^{(\ell)}, \mathbf{b}^{(\ell)}\}_{\ell=1}^{n_l}$.
Construct input vector $\mathbf{x}_p^{(i)} \in \mathbb{R}^n$ for vertex p in walk i, $1 \leq p \leq l$, $1 \leq i \leq |\Omega|$.
while not stopping criterion **do**
    Perform a feedforward pass to compute $f^{(\ell)}(\mathbf{x}_p^{(i)})$.
    For the output layer n_l, set $\delta^{(n_l)}$ using Eq. (6.8)
    for $\ell = n_l - 1, n_l - 2, n_l - 3, \ldots, 1$ **do**
        Compute "error terms" $\delta(\ell)$ using Eq. (6.9).
        if $\ell > \frac{n_l}{2}$ **then**
            Compute $\nabla_{\mathbf{W}^{(\ell)}} J(\mathbf{W}, \mathbf{b})$ and $\nabla_{\mathbf{b}^{(\ell)}} J(\mathbf{W}, \mathbf{b})$ using Eq. (6.10)-(6.11).
        else
            Compute $\nabla_{\mathbf{W}^{(\ell)}} J(\mathbf{W}, \mathbf{b})$ and $\nabla_{\mathbf{b}^{(\ell)}} J(\mathbf{W}, \mathbf{b})$ using Eq. (6.12)-(6.13).
        end if
        Determine the step size ξ by line search.
        Update $\mathbf{W}^{(\ell)} = \mathbf{W}^{(\ell)} - \xi \nabla_{\mathbf{W}^{(\ell)}} J(\mathbf{W}, \mathbf{b})$.
        Update $\mathbf{b}^{(\ell)} = \mathbf{b}^{(\ell)} - \xi \nabla_{\mathbf{b}^{(\ell)}} J(\mathbf{W}, \mathbf{b})$.
    end for
end while
Compute embedding results $f^{\frac{n_l}{2}}(\mathbf{x}_p^{(i)})$.



(a) the Zachary's karate network (b) Embedding results of NETWALK

Figure 6.3: Embedding results on Zachary's karate network

order to determine whether an incoming edge is an anomaly, we build a lookup table to encode new edge(s) in real-time based on the network representations we have learned. For undirected networks, the operator has to be symmetric. That is, for any edge (u, v) or (v, u), the edge representation should be the same. In this paper, we use the Hadamard operator which has shown good performance in edge encoding [Grover and Leskovec, 2016]. Assume that the d-dimensional representation learned by Algorithm 6 for vertex v is $f(v)$, then the representation of each edge (v, u) under Hadamard operator is $[f(v) \circ f(u)]_i = f_i(v) \times f_i(u)$. It is worth mentioning that the way to encode edges is very flexible. We can add any additional edge-specific features to augment the edge vector.

6.1.2.4 Maintaining Network Representations Incrementally

To cope with the fast evolving nature of dynamic networks, we prefer to update the network representations without having to maintain explicitly the complete details of network structures. This section describes how the network representations learned by NETWALK are dynamically updated upon changes of the network. Each added/deleted edge affects in a number of network walks which will be used to update the current network representation. In our model, we design a reservoir-based algorithm to maintain a compact record which consists of a set of "neighbors" for each vertex, and the walks are updated based on the reservoir for each vertex.

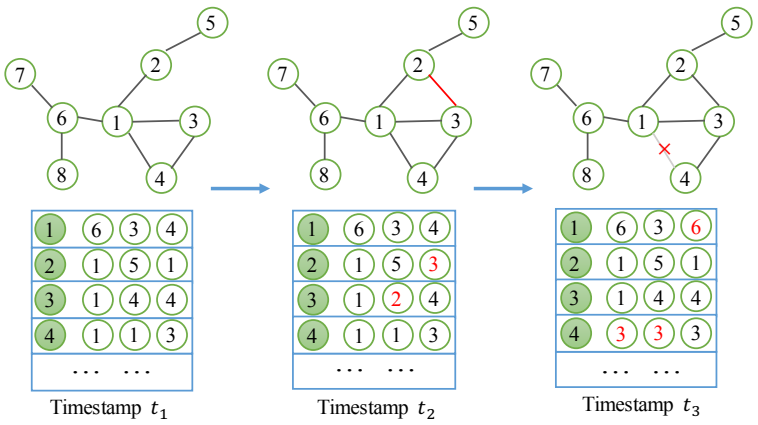


Figure 6.4: Illustration of updating the reservoirs

Definition 4 (Vertex Reservoir)**.** *For each vertex $v \in \mathcal{V}$, the corresponding vertex reservoir S_v is a set of vertex with ψ items, which are sampled with replacement from v's neighbors $ne_v = \{u|(u,v) \in \mathcal{E}, u \neq v\}$.*

Given a stream of edges, NETWALK maintains a reservoir for each vertex v such that each single item in the reservoir is selected at random from v's neighbors. Thus the reservoir needs to be updated as new edges arrive. The updating rules are described as follows for each newly added edge (u, v):

1. update the degree of vertices u and v: $D_{u,u} = D_{u,u} + 1$, $D_{v,v} = D_{v,v} + 1$;
2. for each item in the reservoir S_u, with probability $\frac{1}{D_{u,u}}$, replace the old item with the new item v; and with probability $1 - \frac{1}{D_{u,u}}$, keep the old item;
3. for each item in the reservoir S_v, with probability $\frac{1}{D_{v,v}}$, replace the old item with the new item u; and with probability $1 - \frac{1}{D_{v,v}}$, keep the old item.

Lemma 1. *For each i, the i^{th} neighbor of vertex v is chosen to be included in the reservoir S_v with probability $\frac{\psi}{D_{v,v}}$.*

Proof. We will prove it by induction. After the $(i-1)^{\text{th}}$ round, let us assume that the probability of an item being in the reservoir S_v is $\frac{\psi}{D_{v,v}}$. Since the probability of the item being replaced in the

i^{th} round is $\frac{1}{D_{v,v}+1}$, the probability that a given item is in the reservoir after the $(i-1)^{\text{th}}$ round will be $\frac{\psi}{D_{v,v}} \times (1 - \frac{1}{D_{v,v}+1}) = \frac{\psi}{D_{v,v}+1}$. We update $D_{v,v} \leftarrow D_{v,v} + 1$. Hence, the result holds for i, and is therefore true by induction. □

In case where edges are deleted, the reservoir is chosen similarly to aforementioned rules. In this case, one needs to update the degree matrix first, and then replace the deleted items with the remaining neighbors of the corresponding vertex. As illustrated in Figure 6.4, when (v_2, v_3) is added at timestamp t_2, the corresponding reservoirs of v_2 and v_3 will be updated by adding v_3 with a probability of $\frac{1}{3}$, and v_2 with a probability of $\frac{1}{3}$, respectively. Similarly, when (v_1, v_4) is deleted at timestamp t_3, we replace the deleted item v_1 with v_3 with a probability of 1 (there is only one remaining neighbor of the corresponding vertex v_4), and replace the deleted item v_4 with v_3 or v_6 with probability $\frac{1}{2}$.

After updating the reservoir of the corresponding vertices as edge (u, v) arrives, we will generate the network walks that need to be updated accordingly. For each newly added edge (u, v), the walks need to be added are defined as $\Omega^+ = \{(u_1, u_2, ..., u_i, u, v, v_1, v_2, ..., v_j) \vee (v_1, v_2, ..., v_i, v, u, u_1, u_2, ..., u_j) | i + j = l - 2\}$, which is a collection of network walks with length l including the new edge (u, v). The transition probability of each connected vertex pair (u_m, u_n) is $p(u_m, u_n) = \frac{1}{D_{u_m,u_m}}$. For each edge (u', v') that needs to be removed, the dynamic walk are defined as $\Omega^- = \{\omega | \forall \omega \in \Omega \wedge ((u', v') \in \omega \vee (v', u') \in \omega)\}$. We will then continue to train the model with the updated network walk set in a warm-start fashion. The pseudocode of updating network representations is shown in Algorithm 7.

Discussion. 1). *On the incremental online training*. When new edges come, ideally, we need to retrain the embedding model on the whole walk set $\Omega \cup \Omega^+$. However, this is usually time-consuming. Many online gradient decent methods have been discussed for this problem. For example, we can sample a small set of walks from Ω based on their gradients [Defazio et al., 2014] and add them to Ω^+ for training. For the edge deletion case, we retrain the model with the updated walk set $\Omega - \Omega^-$ for edge deletion. This is time consuming if the original walk set Ω is very large.

Algorithm 7 Network Representation Maintenance

Input: Network walk set Ω, a streaming edge set $E^{(t)}$; saved clique embedding model.

Output: The updated Ω, the updated embedding clique model.

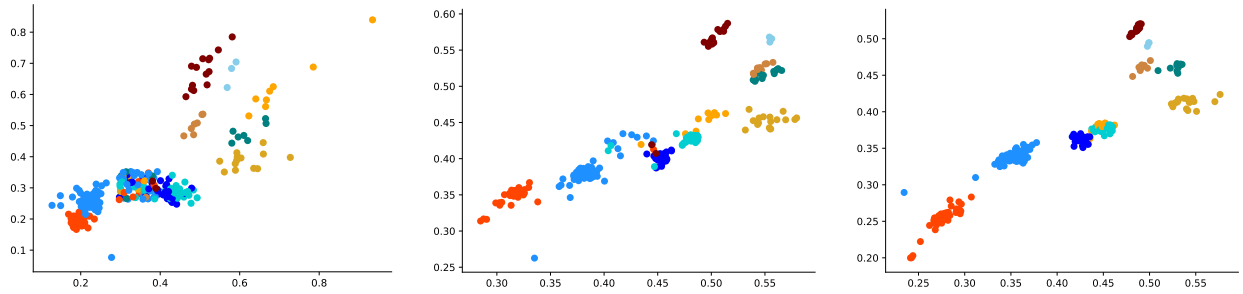
1: **for** (u, v) in the streaming edge set $E^{(t)}$ **do** ▷ dynamic walk generation
2: Update vertex set $\mathcal{V}$.
3: Update degree matrix $\mathbf{D}$.
4: Update the reservoirs S_u and S_v using the rules described in Section 6.1.2.4.
5: Generate the network walk sets Ω^+ for new edges and Ω^- for deleted edges, respectively.
6: **end for**
7: Load the saved embedding model. ▷ model update
8: Train the model with the dynamic network walk set Ω^+, or with the updated walk set $\Omega - \Omega^-$ if with edge deletion.
9: Update network representations $f^{(\frac{n_l}{2})}(\mathbf{x}_p^{(i)})$.
10: Save the updated clique embedding model.

We can also incorporate the edge deletion part into the objective function Eq. (6.7),

$$\begin{aligned} J(\mathbf{W}, \mathbf{b}) = & \sum_{i=1}^{|\Omega^+|} \sum_{1 \le p,q \le l} \left\| f^{(\frac{n_l}{2})}(\mathbf{x}_p^{(i)}) - f^{(\frac{n_l}{2})}(\mathbf{x}_q^{(i)}) \right\|_2^2 \\ & + \sum_{i=1}^{|\Omega^-|} \sum_{1 \le p,q \le l} \max\left(0, \alpha - \left\| f^{(\frac{n_l}{2})}(\mathbf{x}_p^{(i)}) - f^{(\frac{n_l}{2})}(\mathbf{x}_q^{(i)}) \right\|_2^2\right) \\ & + \frac{\gamma}{2} J_{AE} + \beta J_{Sparsity} + \frac{\lambda}{2} J_{Weight\ Decay}, \end{aligned} \tag{6.14}$$

which ensures that the deleted edges in the embedding space have a distance of at least α from each other. In the following evaluation section, we focus on the edge addition scenario which is more common in real world. 2). *On the weighted networks*. Only minor modification on current algorithm is needed to accommodate weighted network anomaly detection. First, since the walks generating step adopts random walker technique, it is easy to consider the weights of edges into the transition probability. Accordingly, in Eq. (6.4), additional weights should be put to the pairwise loss of two vertices.

Visualization. In this subsection, we show the dynamic encoding capability of NETWALK by



(a) initial embedding with 50% edges of the original network

(b) online embedding with additional 25% edges (75% in total)

(c) online embedding with additional 25% edges (100% in total)

Figure 6.5: Embedding results on Email network

applying our method to the Email network[4]. Figure 6.5(a) shows the embedding results with 50% edges, in which the vertex color indicates the community to which each vertex belongs. Figure 6.5(b) presents the online embedding results with additional 25% edges, and Figure 6.5(c) updates the embeddings with the remaining 25% edges. Notably, more and more linearly separable clusters can be found in the 2-dimensional representation space in Figure 6.5(b) and (c).

Computational Analysis. In network walk generation section, the time complexity to generate $|\Omega|$ walks with length l in a network with n vertices is $\mathcal{O}(nl|\Omega|)$. The edge encoding step takes $O(md)$ time to encode m edges with vertex dimension d. For each newly added edge, it takes $O(\psi)$ time to update the corresponding reservoirs, and $O(\psi l)$ time to generate the walks that need to be retrained.

6.1.3 Anomaly Detection

The network representations learned by NETWALK can be beneficial for lots of downstream applications, such as link prediction, anomaly detection and community detection. In this paper, we focus on the anomaly detection problem based on the learned network representations. We define the anomaly detection problem in dynamic network as follows: given the vertex representations $f_{\mathbf{W},\mathbf{b}}(\mathbf{x}_p^{(i)}) \in \mathbb{R}^d$ or corresponding edge representations, group existing representations into k clusters, and detect any newly arriving vertices or edges that do not naturally belong to any existing

[4]https://snap.stanford.edu/data/email-Eu-core-temporal.html

cluster. This may include the following scenarios: 1) the vertex or edge corresponds to an anomaly; 2) the vertex or edge marks the start of a new cluster in the network stream. It is difficult to distinguish these two cases unless we receive more streaming data afterwards. So in our model, we find the closest cluster to each point. We use the Euclidean distance as the similarity measure, given by $||\boldsymbol{c} - f(\cdot)||_2$, where $\boldsymbol{c}$ is the cluster center and $f(\cdot)$ is the learned representation for each vertex or edge. The anomaly score for each point is reported as its closest distance to any cluster centers.

When new edges stream in, we need to update cluster centers accordingly. In this paper, we leverage the streaming k-means clustering [Ailon et al., 2009] which uses parameters to control the decay of estimates. Here, we introduce a decay factor α when calculating the new cluster centers after absorbing new point(s). We use the parameter α to control the importance of "older" data points in existing clusters. Assuming that there are n_0 points $\{\boldsymbol{x}_i\}_{i=1}^{n_0}$ in an existing cluster and n' new points $\{\boldsymbol{x}'_i\}_{i=1}^{n'}$ at time-stamp T' to be absorbed by this cluster, the centroid $\boldsymbol{c}$ can be updated in the following way

$$\boldsymbol{c} = \frac{\alpha \boldsymbol{c}_0 n_0 + (1-\alpha)\sum_{i=1}^{n'} \boldsymbol{x}'_i}{\alpha n_0 + (1-\alpha)n'}, \tag{6.15}$$

where $\boldsymbol{c}_0$ is the previous cluster center. The decay factor α is chosen as 0.5 and used to ignore older instances, which is analogous to an exponentially-weighted moving average.

Computational Analysis. With k clusters signified by k center vectors, finding the nearest cluster takes only $O(kd)$ time. It takes $O(d)$ time to compute the anomaly score for each data point. Updating the centers takes $O(d)$ time with respect to the dimension of vertex/edge representations. Thus the total time complexity of anomaly detection is $O(kd)$ for each incoming data point.

6.1.4 Evaluation

To verify the performance of the proposed NETWALK model, we conduct experiments on a variety of dynamic networks from different domains including *UCI Msg* [Opsahl and Panzarasa, 2009], *Digg*, *arXiv hep-th* [Leskovec et al., 2007] and *DBLP*. Detailed descriptions of the datasets can be found in Section A.1. The competing methods used here including four network embedding baselines, namely Spectral Clustering (SC) [**?**], DeepWalk [Perozzi et al., 2014b], node2vec [Grover and Leskovec, 20 and SDNE [Wang et al., 2016], and two streaming anomaly detection baselines GOutlier [Aggarwal et al., 2011]

Table 6.1: Anomaly detection performance comparison

Methods	UCI Msg			arXiv hep-th			Digg			DBLP		
	1%	5%	10%	1%	5%	10%	1%	5%	10%	1%	5%	10%
GOutlier	0.7181	0.7053	0.6707	0.6964	0.6813	0.6322	0.6963	0.6763	0.6353	0.7172	0.6891	0.6460
CM-Sketch	0.7270	0.7086	0.6861	0.7030	0.6709	0.6386	0.6871	0.6581	0.6179	0.7097	0.6892	0.6332
SC	0.6324	0.6104	0.5794	0.6114	0.6034	0.5593	0.5949	0.5823	0.5591	0.6141	0.6245	0.5915
DeepWalk	0.7514	0.7391	0.6979	0.7312	0.7000	0.6644	0.7080	0.6881	0.6396	0.7413	0.7202	0.6657
node2vec	0.7371	0.7433	0.6960	0.7374	0.7137	0.6748	0.7364	0.7081	0.6508	0.7368	0.7193	0.6786
SDNE	0.7307	0.7144	0.6868	0.7221	0.7041	0.6609	0.7160	0.6804	0.6340	0.7342	0.7160	0.6565
NetWalk	**0.7758**	**0.7647**	**0.7226**	**0.7489**	**0.7293**	**0.6939**	**0.7563**	**0.7176**	**0.6837**	**0.7654**	**0.7388**	**0.6858**

and CM-Sketch [Ranshous et al., 2016].

6.1.4.1 Identifying Anomalies

In this section, we evaluate NETWALK in two settings: static and streaming. In the static setting, the first 50% edges of the network is used for training, and the rest incoming edges are used for testing. The vertex representations are learned offline. We then use the representations to encode and cluster the training edges. The test edges are scored and ranked based on their distances to the closest cluster centers. The goal is to quantify the effectiveness of network representations of NETWALK in the anomaly detection task. Due to the challenges in collecting data with ground-truth anomalies, we use anomaly injection method to create the anomalous edges [Akoglu et al., 2015].

The area under curve (AUC) score is used to measure the predictive power of all methods. We rank and score all encoded testing edges by calculating the distance to the closest center in the cluster generated from the training edges based on their representations, as presented in Table 6.1. The parameters of NETWALK are tuned by 5-fold cross-validation on a rolling basis using the initial network. Here we set n_l, the layers of the autoencoder network, to 6. The latent dimension of vertex representation is set to 200, 200, 20 for each encoding layer. The walk length l is set to 3. The number of samples per vertex is $\psi = 20$. The other parameters are chosen as follows: the weight of reconstruction constraint $\gamma = 5$, sparsity ratio $\rho = 0.1$, the weight of sparsity $\beta = 0.2$, weight decay term $\lambda = 5e{-}4$, number of clusters $k = 10$. The maximum iteration of NETWALK is set to 500. For the first four network embedding methods SC, DeepWalk, node2vec and SDNE, we

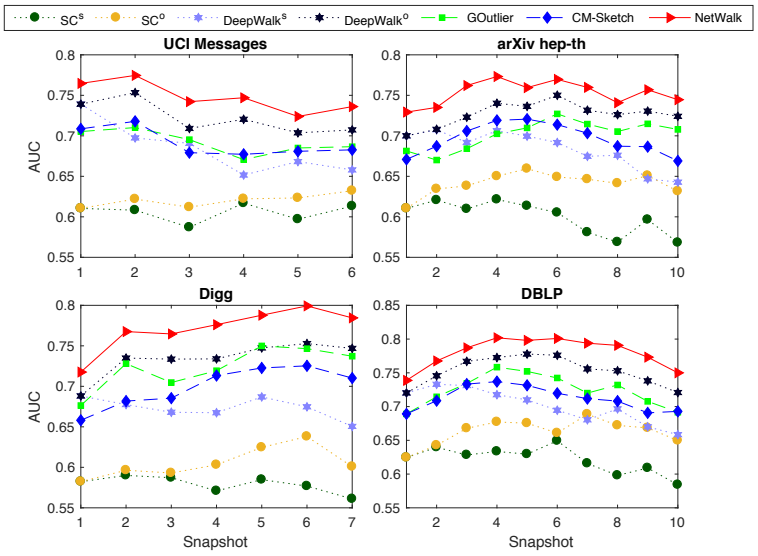


Figure 6.6: Accuracy of anomaly detection on dynamic network with 5% anomalies

use the same clustering and ranking method for anomaly detection based on the learned representations. The testing edges of all datasets are injected 1%, 5% and 10% anomalies, respectively. It is evident from Table 6.1 that, 1) network embedding-based approaches (e.g. DeepWalk, node2vec and SDNE) outperform traditional sketch-based models (GOutlier and CM-Sketch), 2) NETWALK obtains a higher AUC than other baselines on all datasets. And even if 10% anomalies are injected, the performance of NETWALK is still acceptable.

In the streaming setting, we again use the first 50% edges for training to build the initial network representations and clusters. The testing edges arrive sequentially and are processed online. In other words, all testing edges are only partially visible at any given time. For convenience of comparison, we split the streaming edges into several snapshots. The number of edges for each snapshot are set to $1k$, $10k$, $6k$, $30k$ respectively for different dataset based on their test set sizes. For each arriving snapshot, NETWALK updates the corresponding network representations, clusters, and anomaly scores. For the network embedding baselines, we only include SC and DeepWalk since the performance of node2vec and SDNE are close to DeepWalk. These two embedding baselines are designed for static network (DeepWalk has to generate the new random walks based on the entire network), thus we adopt two versions in the evaluation. 1)The static version SC^s and $DeepWalk^s$: the latent vertex representations are learned only based on the initial network,

and there are no updates upon receiving new edges; 2) The online version SC^o and DeepWalk^o: the algorithm is repeated using all previous $t-1$ snapshots and tested with the t^{th} snapshot. The anomaly percentage for all datasets is set to 5%. Other parameters are chosen similarly as mentioned above. The accuracies are reported in Figure 6.6. We observe that 1) the online versions SC^o and DeepWalk^o achieve better accuracy than the corresponding static ones, 2) NETWALK outperforms other baselines by a large margin. Note that the online version of DeepWalk needs to store the entire network in memory and repeats the walk generation at each snapshot. So our method is much more efficient on this aspect (see Section 6.1.4.2).

6.1.4.2 Dynamic Maintenance Performance

In this section we will show that the proposed NETWALK delivers both accurate and efficient solution in a streaming setting on *UCI Msg*. Note that NETWALK leverages the reservoir sampling technique to maintain network representations incrementally. Specifically, NETWALK creates a reservoir for each vertex which contains its neighbors, and new network walks are generated based on these reservoirs without storing the entire network in memory. In this part, we test the performance of dynamic representation maintenance by comparing it with a version of NETWALK^s which needs to keep the network in memory and generate walks based the entire network at the current timestamp on *UCI Msg* with 5% anomalies.

It can be seen from Figure 6.7 that the average AUC score of NETWALK is 0.7329 which is higher than NETWALK^s (0.7307). However, the walk generation time of NETWALK^s is 5.0 to 10.2 times longer than NETWALK. Similar to NETWALK^s, DeepWalk and node2vec need to keep the entire network in memory to update the walks. Therefore, DeepWalk and node2vec take longer time ($5\times$ to $11\times$) to generate walks than our model. As for SDNE, the calculation of the first-order proximity on the network level is very time consuming compared with other baselines.

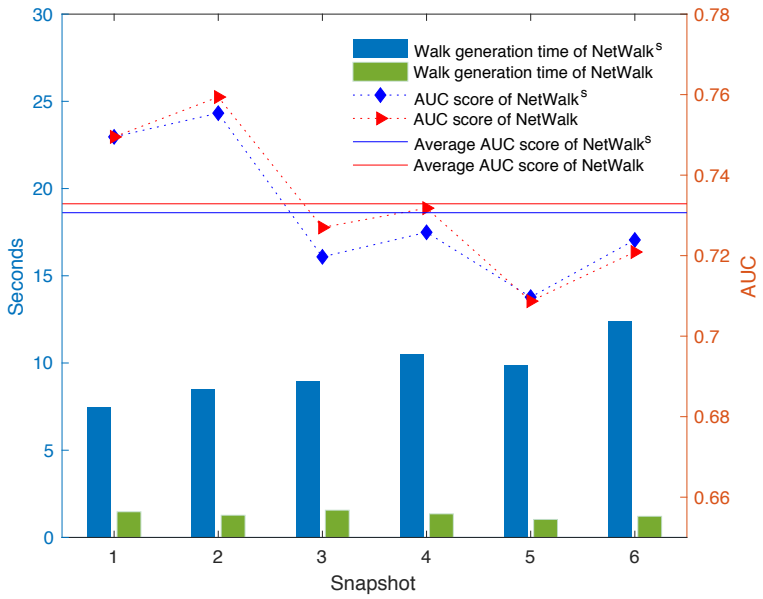


Figure 6.7: Dynamic maintenance performance evaluation on UCI Messages dataset

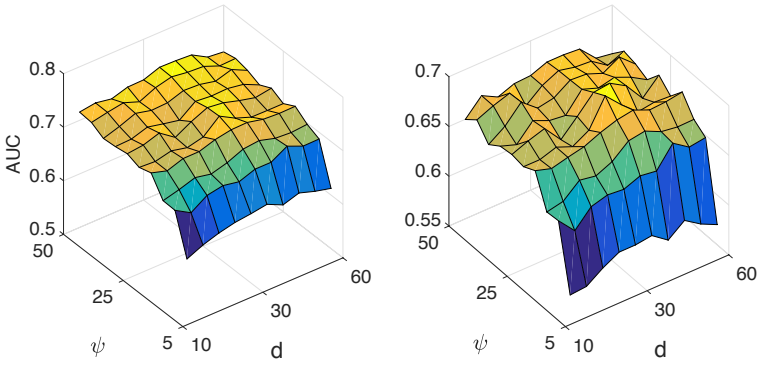


Figure 6.8: Anomaly detection performance of *UCI Msg* and *Digg* with different parameter pairs

6.1.4.3 Parameter Sensitivity

The NETWALK framework involves a number of parameters that may affect its performance. We examine such changes in performance on two anomaly detection tasks (`UCI Messages` and `Digg`). We vary the number of samples per vertex (ψ), the dimensions of vertex representation (d), the training data percentage and the walk length (l) to determine their impacts on anomaly detection. Except for the parameters being tested, all other parameters assume default values.

We first examine different choices of parameters ψ and d. We choose values of ψ from 5 to 50, and let d vary from 10 to 60 with an interval 10. The results are summarized in Figure 6.8. It is

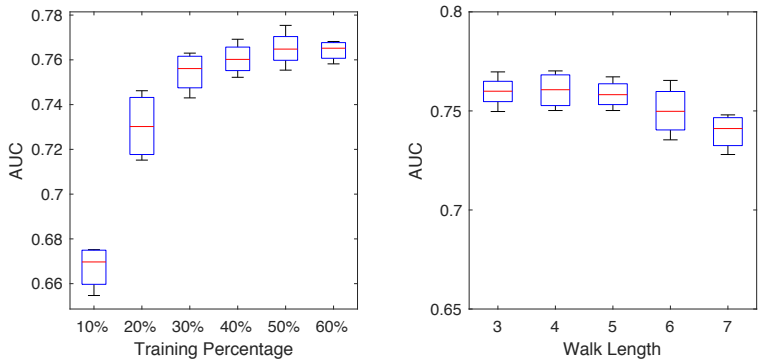


Figure 6.9: Stability over the training percentage of the initial network and the walk length

evident that AUC initially improves with ψ but further improvements are slower beyond a certain threshold ($\psi > 20$). It indicates that NETWALK is able to learn meaningful vertex representations with a small number of network walks. The performance is relatively stable across different values of d. For both datasets, the AUC increases slightly as d increases, and then drops after d reaches 40. It is because, when d is small, information from the input data may be partially missing in the representation learning phase; while when d is too high, the performance of the clustering phase will be weakened.

The training percentage of the initial network and the length of network walk are also important in the anomaly detection task. Figure 6.9 examines the effects of varying the training percentage and the length of network walk. We observe from Figure 6.9(a) that the AUC increases sharply when the training percentage of network goes from 10% to 30%, and then the performance stays relatively stable. It demonstrates that our model can learn a better representation even trained with a small number of data instances. The performance slightly increases when l goes from 3 to 4; after that, the AUC decreases. This is because the clique constraint in Eq. (6.7) forces all the vertices in the same walk to have similar representations, which is too restrictive for longer walks. Taking both prediction performance and computational time into consideration, we will choose a relative small walk length.

6.2 Link Prediction with Spatial and Temporal Consistency

Evolutionary network analysis [Aggarwal and Subbian, 2014] has become increasingly important in recent years. One of the major tasks is temporal link prediction which is to predict the future network structure based on a sequence of observed networks. Formally, in this paper, the problem of temporal link prediction is defined as: given a sequence of networks from timestamps 1 through T, the task is to predict the link weights at timestamp $T + \alpha$, where $\alpha \geq 1$. Note that a special case of this definition is to predict whether a new link will emerge or not (when we restrict the link weight to be either 0 or 1).

Extensive research efforts have been devoted to the temporal link prediction task including non-parametric [Sarkar et al., 2012] and parametric [Tylenda et al., 2009, Oyama et al., 2011] models. The key factors to link prediction task in temporal networks are spatial and temporal consistencies, which mean: 1) a node has higher probability to form a link with a nearby node than with a remote node in the near future; 2) a network usually evolves smoothly over time. The first factor is akin to that in static network link prediction [Lü and Zhou, 2011, Al Hasan and Zaki, 2011]. It encodes the *local* network propagation constraints from the network topology at each timestamp. The second one *globally* enforces the *smoothness* of network evolution over time [Yu et al., 2017b, Aggarwal and Subbian, 2014].

Existing approaches, however, seldom unify these two factors to strive for the spatial and temporal consistency in the dynamic network. Their prediction performance thus degrades. Moreover, most link prediction algorithms focus on the very next timestamp (i.e., $\alpha = 1$), and are unable to predict when $\alpha > 1$. To address the limitation, we propose LIST, a model for link prediction with spatial and temporal consistency. We leverage the time-dependent matrix factorization technique, which has shown to be a powerful means to characterizing dynamic structural data [Koren, 2010, Yu et al., 2017b], to decompose the network adjacency matrices into time-dependent matrices that capture the features of vertices in the dynamic networks. At the same time, we introduce the network propagation constraint [Zhou et al., 2003, Kashima et al., 2009, Cheng et al., 2016] which ensures vertices to be within close proximity to their neighbors in the hidden feature space to be learned by the time-dependent matrix factorization. As depicted in Figure 6.10, we learn the fea-

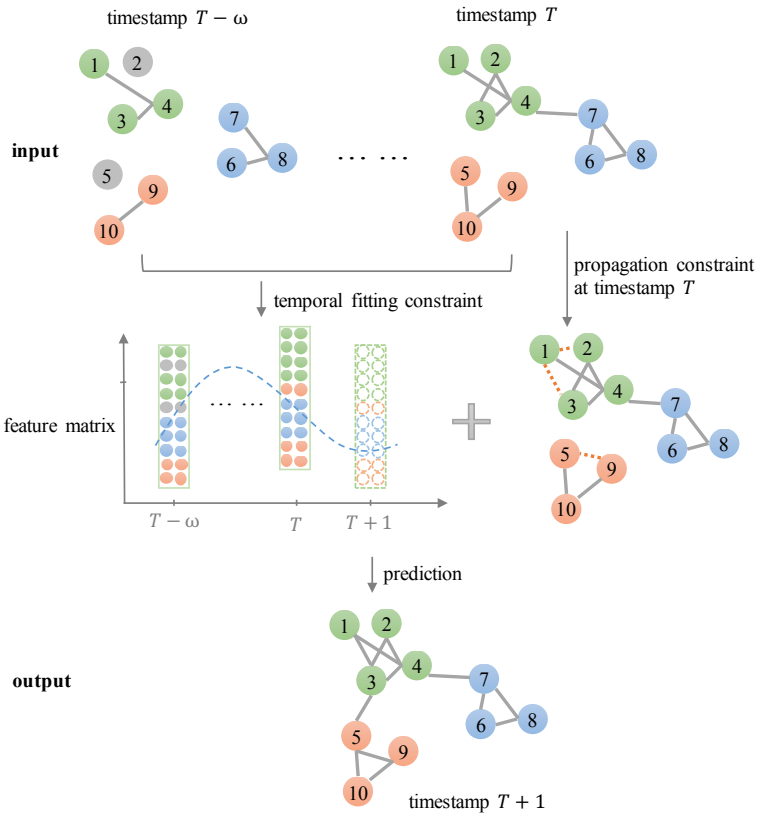


Figure 6.10: Illustration of the LIST model

ture vector of each vertex by simultaneously optimizing the temporal fitting constraint and network propagation constraint. The temporal fitting constraint can be expressed as time-dependent matrix factorization with network adjacency matrix, while the propagation constraint preserves similarities between the connected pairs of vertices in the feature space. The learned feature matrices are parameterized with time and can be used to reconstruct the network structure at any given timestamp t. This allows us to make far more general predictions. Additionally, this feature matrix can be viewed as a complete profile of the network dynamics over time, which may also find its utility in other application settings of evolutionary network analysis.

6.2.1 The LIST Model

Let the observed sequence of temporal networks be $G(t) = (\mathcal{N}, \mathbf{A}(t))$, where $\mathcal{N}$ is a set of vertices, and $\mathbf{A}(t)$ is the adjacency matrix of the network at timestamp $t \in [1, T]$, which is defined as a function of time. We assume that the size of vertex set $|\mathcal{N}| = n$, therefore $\mathbf{A}(t) \in \mathbb{R}^{n \times n}$. The

elements a_{ijt} of $\mathbf{A}(t)$ is the link weight between vertices i and j at timestamp t. Our goal is to predict the links at timestamp $T + \alpha$ given $\mathbf{A}(1), \mathbf{A}(2), ..., \mathbf{A}(T)$.

We adopt the label propagation principle [Zhou et al., 2003, Cheng et al., 2016] which states that two vertices similar to each other are likely to have the same label. We consider a practical assumption that *two vertices that are connected are likely to have similar features*. Under this assumption, each vertex adjusts its feature vector based on its neighbors. Suppose that the initial feature vector of vertex i is $\mathbf{v}_i(t)$ and the final state is $\mathbf{f}_i(t)$ at timestamp t. Then the propagation process from $\mathbf{v}_i(t)$ to $\mathbf{f}_i(t)$ can be modeled by the following optimization problem.

$$\begin{aligned}\min_{\mathbf{f}(t)} \quad & \lambda \sum_{i,j} \mathbf{A}_{ij}(t) \| \frac{1}{\sqrt{\mathbf{D}_{ii}(t)}} \mathbf{f}_i(t) - \frac{1}{\sqrt{\mathbf{D}_{jj}(t)}} \mathbf{f}_j(t) \|_2^2 \\ & + (1-\lambda) \sum_i \|\mathbf{f}_i(t) - \mathbf{v}_i(t)\|^2 \end{aligned} \tag{6.16}$$

where $\mathbf{D}(t) \in \mathbb{R}^{n \times n}$ is the degree matrix of $\mathbf{A}(t)$. $\lambda \in (0, 1)$ is the regularization weight. The first term is the smoothness constraint, which enforces the neighboring vertices to have similar feature vectors. The second term is the fitting constraint, which penalizes large deviation from the initial feature vectors. The analytical solution of Eq. (6.16) is:

$$\mathbf{f}_i(t) = (1-\lambda)(\mathbf{I} - \lambda \tilde{\mathbf{A}}(t))^{-1} \mathbf{v}_i(t) \tag{6.17}$$

where $\mathbf{I} \in \mathbb{R}^{n \times n}$ is the identity matrix. $\tilde{\mathbf{A}}(t)$ is the normalized version of $\mathbf{A}(t)$ which is defined as $\sqrt{\mathbf{D}(t)}\mathbf{A}(t)\sqrt{\mathbf{D}(t)}$. This explicit solution shows that the final feature vector is a transformation of the initial one based on the network structure at timestamp t.

Then how do we determine $\mathbf{f}_i(t)$? In this paper we leverage the time-dependent matrix factorization method which naturally expresses the evolving network by learning a low rank representation of the underlying adjacency matrix. Let's focus on undirected networks for now. In this case, the symmetric adjacency matrix $\mathbf{A}(t)$ can be reconstructed by the feature vectors $\{\mathbf{f}_i(t)\}_{i=1}^n$,

$$\mathbf{A}(t) = \mathbf{F}(t)\mathbf{F}(t)^\top \tag{6.18}$$

Here $\mathbf{F}(t) = [\mathbf{f}_1, \mathbf{f}_2, ..., \mathbf{f}_n] \in \mathbb{R}^{n \times k}$ is a time-depend feature matrix.

We then follow a standard approach to set up a least squares optimization problem so that $\mathbf{A}(t)$ and $\mathbf{F}(t)\mathbf{F}(t)^\top$ are as close as possible. This can be achieved by minimizing the Euclidean distance

between all entries in $\mathbf{A}(t)$ and $\mathbf{F}(t)\mathbf{F}(t)^\top$. Therefore, the optimization problem can be written as,

$$\min_{\mathbf{F}} \sum_{t=\max(1,T-\omega)}^{T} \frac{h(t)}{2} \|\mathbf{A}(t) - \mathbf{F}(t)\mathbf{F}(t)^\top\|_F^2 \tag{6.19}$$

Here, $h(t) = e^{-\theta(T-t)}$ is an exponential decay function with time t that regulates the importance of the current timestamp of the network with respect to the past timestamps. ω is the sliding window size which only takes the recent ω timestamps into consideration. This is pragmatic because there is no need to store all network timestamps.

As it can be seen from Eq. (6.19), in order to learn $\mathbf{F}(t)$, one has to regulate the time-dependent form of this matrix. Let $\mathbf{P}(t) = (1-\lambda)(\mathbf{I} - \lambda\tilde{\mathbf{A}}(t))^{-1}$, we have $\mathbf{F}(t) = \mathbf{P}(t)\mathbf{V}(t)$ based on Eq. (6.17). Then the optimization problem described by Eq. (6.19) has the following form,

$$\min \sum_{t=\max(1,T-\omega)}^{T} \frac{h(t)}{2} \|\mathbf{A}(t) - \mathbf{P}(t)\mathbf{V}(t)\mathbf{V}(t)^\top\mathbf{P}(t)^\top\|_F^2 \tag{6.20}$$

Here $\mathbf{V}(t) \in \mathbb{R}^{n\times k}$ is a time-dependent matrix including all initial states of vertex feature vectors. It characterizes the network dynamics by modeling the changes in the vertex feature space. The function $\mathbf{V}(t)$ can take on any canonical form, such as linear models, polynomial models etc. based on the specific tasks. We choose polynomial function for $\mathbf{V}(t)$ because we are trying to fit the network dynamics within a small sliding window of length ω [**?**]. Thus $\mathbf{V}(t)$ can be represented as follows:

$$\mathbf{V}(t) = \mathbf{W}^{(0)} + \mathbf{W}^{(1)}t + ... + \mathbf{W}^{(d)}t^d = \sum_{i=0}^{d} \mathbf{W}^{(i)}t^i \tag{6.21}$$

Here $\{\mathbf{W}^{(i)}\}_{i=0}^{d} \in \mathbb{R}^{n\times k}$, $d \in \mathbb{N}_+$. $\mathbf{V}(t)$ is the simple linear function if $d = 1$.

The challenge of optimizing the objective function defined in Eq. (6.20) is that the network $\mathbf{A}(t)$ could be very large and sparse, and the optimization of a $O(n^2)$ objective function is often too computationally expensive. Generally, the existence of a link provides more information than the absence of a link which conveys far more noises. Therefore, Eq. (6.20) should be tuned up to focus on the nonzero entries in the adjacency matrix $\mathbf{A}(t)$. Let m be the number of nonzero entries in $\mathbf{A}(t)$. However we still need a sample of zero entries to properly train the model. In this paper, the sample size is set to be equal to the size of nonzero entries m. Let $S(t)$ be the sample indices at timestamp t such that $a_{ijt} = 0, \forall(i,j) \in S(t)$. Let $E(t)$ be the set of indices that need

to be optimize in Eq. (6.20), then we have $E(t) = \{(i,j)|a_{ijt} > 0\} \cup S(t)$. Note that the size of $E(t)$ is much smaller than $O(n^2)$ because the networks are often very sparse in practice. Then the aforementioned objective function can be presented as follows,

$$\sum_{t=\max(1,T-\omega)}^{T} \frac{h(t)}{2} \sum_{(i,j)\in E(t)} (a_{ijt} - (\mathbf{P}(t)\mathbf{V}(t)\mathbf{V}(t)^\top \mathbf{P}(t)^\top)_{ij})^2 \tag{6.22}$$

For directed networks, $\mathbf{A}(t)$ can be decomposed into two matrices: a constant matrix $\mathbf{U}$ and a time-dependent matrix $\mathbf{V}(t)$, both are $n \times k$ matrices. In this case, the optimization problem comes to be,

$$\sum_{t=\max(1,T-\omega)}^{T} \frac{h(t)}{2} \sum_{(i,j)\in E(t)} (a_{ijt} - (\mathbf{U}\mathbf{V}(t)^\top \mathbf{P}(t)^\top)_{ij})^2 \tag{6.23}$$

Note that one can choose both $\mathbf{U}$ and $\mathbf{V}(t)$ to be time-dependent, but it will double the parameter space and increase the model complexity. Similar results can be achieved if one choose to make $\mathbf{U}$ time-dependent rather than $\mathbf{V}(t)$. In the empirical study section, we focus on the undirected networks.

6.2.1.1 Model Learning

In this section, we describe the algorithm to learn the LIST model. The aforementioned objective function depends on the number of links in the networks which is easy to compute. We also need to add weight-decay terms to reduce the variance of our model. Consider the objective function $J(\mathbf{W})$ for undirected networks with weight-decay terms,

$$\sum_{t=\max(1,T-\omega)}^{T} \frac{h(t)}{2} \|\mathbf{1}_{E(t)}(\mathbf{A}(t) - \mathbf{P}(t)\mathbf{V}(t)\mathbf{V}(t)^\top \mathbf{P}(t)^\top)\|_F^2 + \sum_{i=0}^{d} \frac{\beta_i}{2} \|\mathbf{W}^{(i)}\|_F^2 \tag{6.24}$$

where $\mathbf{1}_{E(t)}(\mathbf{M}) = \begin{cases} \mathbf{M}_{ij} & \text{if } (i,j) \in E(t) \\ 0 & \text{if } (i,j) \notin E(t) \end{cases}$, $\{\beta_i\}_{i=0}^d$ are the weights. In order to infer the parameter $\mathbf{W}$, we leverage the symmetric matrix factorization technique [Kuang et al., 2012] to compute the derivatives of Eq. (6.24). We introduce an "error term" $\psi(t)$ for each timestamp t of undirected networks as follows:

$$\psi(t) = \mathbf{1}_{E(t)}(\mathbf{A}(t) - \mathbf{P}(t)\mathbf{V}(t)\mathbf{V}(t)^\top \mathbf{P}(t)^\top) \tag{6.25}$$

Note that the error matrix in $\psi(t)$ has already projected to indices set defined by $E(t)$. Consider the derivative calculation of a simplified loss function $J = \frac{1}{2}\|\mathbf{A} - \mathbf{P}\mathbf{V}\mathbf{V}^\top\mathbf{P}^\top\|_F^2$ without timestamp t,

$$\begin{aligned}\frac{\partial J}{\partial \mathbf{V}} =& -\mathbf{P}^\top\mathbf{A}\mathbf{P}\mathbf{V} - \mathbf{P}^\top\mathbf{A}^\top\mathbf{P}\mathbf{V} + 2\mathbf{P}^\top\mathbf{P}\mathbf{V}\mathbf{V}^\top\mathbf{P}^\top\mathbf{P}\mathbf{V} \\ =& -\mathbf{P}^\top((\mathbf{A} - \mathbf{P}\mathbf{V}\mathbf{V}^\top\mathbf{P}^\top) + (\mathbf{A} - \mathbf{P}\mathbf{V}\mathbf{V}^\top\mathbf{P}^\top)^\top)\mathbf{P}\mathbf{V}\end{aligned}$$

Here we extract the common factor $\mathbf{A} - \mathbf{P}\mathbf{V}\mathbf{V}^\top\mathbf{P}^\top$ for each term in J. The reason is that we can apply projection $\mathbf{1}_{E(t)}(\cdot)$ on this common factor. So that,

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}^{(i)}} = \frac{\partial J(\mathbf{W})}{\partial \mathbf{V}(t)}\frac{\partial \mathbf{V}(t)}{\partial \mathbf{W}^{(i)}} = \sum_{t=\max(1,T-\omega)}^{T} h(t)\mathbf{P}(t)^\top\left(-\psi(t) - \psi(t)^\top\right)\mathbf{P}(t)\mathbf{V}(t)t^i + \beta_i\mathbf{W}^{(i)} \tag{6.26}$$

We now have the derivative over $\mathbf{W}$ needed to run gradient descent. The pseudocode for LIST model is presented in Algorithm 8.

Algorithm 8 Algorithm for LIST model

Input: temporal adjacency matrices $\{\mathbf{A}(t)\}_{t=\max(1,T-\omega)}^{T}$, the order d of $\mathbf{V}(t)$, latent dimension k.

Output: factor matrices $\{\mathbf{W}^{(i)}\}_{i=1}^{d}$ and the prediction $\mathbf{A}(T+1)$.

Set k, d and ω.
Randomly initialize $\{\mathbf{W}^{(i)}\}_{i=0}^{d}$.
while not stopping criterion **do**
    Compute "error term" $\psi(t)$ for each time stamp t.
    Compute partial derivatives $\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}^{(i)}}$ using $\psi(t)$ by Eq. (6.26).
    Determine the step size λ by line search.
    for i in $\{1, ..., d\}$ **do**
        Update $\mathbf{W}^{(i)} = \mathbf{W}^{(i)} - \lambda\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}^{(i)}}$.
        Compute prediction result $\mathbf{A}(T+1) = (\sum_{i=0}^{d}\mathbf{W}^{(i)}(T+1)^i)(\sum_{i=0}^{d}\mathbf{W}^{(i)}(T+1)^i)^\top$.
    end for
end while

6.2.1.2 Computational Speed-up

Observe that the update of $\mathbf{W}^{(i)}$ in Algorithm 8 has to compute the inverse of an $n \times n$ matrix $\mathbf{I} - \lambda\tilde{\mathbf{A}}(t)$ which runs in time $O(n^3)$. In this section, an iterative method is used to approximate the matrix inverse calculation.

Theorem 4. *Given that the eigenvalues of* $\mathbf{Q} \in \mathbb{R}^{n\times n}$ *in* $[-1, 1]$, $0 < \gamma < 1$, *and the iteration number* B, *the inverse of matrix* $\mathbf{I} - \gamma\mathbf{Q}$ *can be approximated by summing up* $(\gamma\mathbf{Q})^b$ *across all iterations [Zhou et al., 2003], which is*

$$(\mathbf{I} - \gamma\mathbf{Q})^{-1} = \lim_{B\to\infty} \sum_{b=1}^{B} (\gamma\mathbf{Q})^{b-1} \tag{6.27}$$

Given $\lambda \in (0, 1)$ and $\tilde{\mathbf{A}}(t)$ (normalized by degree matrix $\mathbf{D}(t)$), the approximate solution for $\mathbf{P}(t)$ is,

$$\mathbf{P}(t) = (1 - \lambda)(\mathbf{I} - \lambda\tilde{\mathbf{A}}(t))^{-1} = (1 - \lambda) \sum_{b=1}^{B} (\lambda\tilde{\mathbf{A}}(t))^{b-1} \tag{6.28}$$

Assume that the number of nonzero entries in the sparse matrix $\tilde{\mathbf{A}}(t)$ is m. The complexity upper-bound of sparse matrix multiplication is $O(mn)$ [Yuster and Zwick, 2005]. Therefore the above solution takes $O((b - 1)mn)$ in time for each iteration, $b \in [1, B]$, which results in an overall complexity $O(B^2mn)$ for the calculation of $\mathbf{P}(t)$. If we cache the computed results in previously iteration, then the time complexity is reduced to $O(Bmn)$.

6.2.1.3 Complexity Analysis

We assume that the gradient-descent method in Algorithm 8 is implemented for M iterations, and the rank of the factorization is k. The bottleneck step is to update all parameters for $d + 1$ matrices in each of these iterations. It can be seen from Eq. (6.26) that there are ω timestamps within each sliding window, and in each timestamp the complexity for matrix manipulation is $O(Bmn) + O(n^2k)$. Thus, the asymptotic running time is $O(M\omega(d + 1)(Bmn + n^2k))$. In empirical study settings, M, ω, d, B and k are much smaller than n and m, therefore the time complexity is approximately $O(mn + n^2)$.

6.2.2 Evaluation

The ability to predict link weights at a specific time is almost trivial using the LIST model. Once we obtain the factor matrices $\{\mathbf{W}^{(i)}\}_{i=1}^{d}$, the *predicted* structure of the network can be effectively reconstruct by $\mathbf{V}(T+\alpha)\mathbf{V}(T+\alpha)^{\top}$ for any $\alpha \geq 1$. Of course, as the value of α becomes larger and larger, one can expect the reconstruction to become increasingly challenging. In this section, we will compare with the baselines on single-timestamp link weight prediction ($\alpha = 1$), and show the advantages of the LIST model on multiple-timestamp link weight prediction ($\alpha > 1$).

To verify the performance of the LIST model, we conduct experiments on four dynamic networks, namely *Infectious* [Isella et al., 2011], *UCI Msg* [Opsahl and Panzarasa, 2009], *Digg*[5] and *DBLP*, as described in Section A.1. For comparison, we consider the canonical link prediction method Weighted Common Neighbors (WCN) [Zhao et al., 2015], as well as recent algorithms High-performance Link Prediction (HPLP) [Lichtenwalter et al., 2010], CP Tensor Model (CP-Tensor) [Dunlavy et al., 2011] and Temporal Matrix Factorization (TMF) [Yu et al., 2017b]. Note that the HPLP algorithm used here is a modified version that trains a regression model to predict the link weights. We analyze all algorithms by measuring the accuracy of link weight prediction based on root mean-squared error (RMSE).

6.2.2.1 Single-timestamp Link Weight Prediction

To compare the performance of single-timestamp link weight prediction ($\alpha = 1$), we utilize the network timestamps from $T - \omega$ to $T - 1$ as the training set, and the T^{th} timestamp as the test set, $T \in [2, \mathcal{T}]$, where $\mathcal{T}$ is the total number of timestamps in each dataset. We set the order d of time-dependent matrix $\mathbf{V}(t)$ to 1. As reported in [Yu et al., 2017b], hight order of d may slightly improve the performance, but not always. The other parameter settings are as follows: iteration number $B = 100$ for the computation of $\mathbf{P}(t)$, latent dimension $k = 20$, exponential decay $\theta = 0.3$, sliding window size $\omega = 5$, propagation balancing weight $\lambda = 0.3$, regularizer weights $\beta_i = 0.01$. The maximum number of iterations of the LIST model is set to 200. We analyze the algorithms by measuring the prediction RMSE at different timestamps as shown in Figure 6.11.

[5] http://konect.uni-koblenz.de/networks

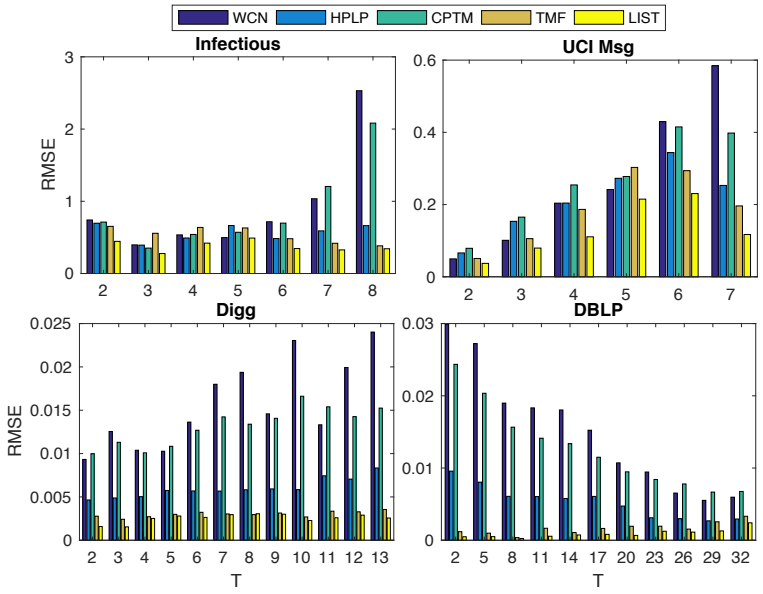


Figure 6.11: Prediction RMSE at timestamp T while training with previous ω timestamps

Table 6.2: Average RMSE across all timestamps

Method	Infectious	UCI Msg	Digg	DBLP
WCN	0.9719±0.8300	0.2719±0.2064	0.0161±0.0048	0.0160±0.0104
HPLP	0.5883±0.6767	0.2702±0.2034	0.0064±0.0012	0.0056±0.0147
CP-Tensor	0.8847±0.9367	0.2196±0.2469	0.0133±0.0136	0.0128±0.0099
TMF	0.5309±0.1185	0.1840±0.1311	0.0032±0.0011	0.0017±0.0028
LIST	**0.3824**±0.1114	**0.1345**±0.0930	**0.0026**±0.0005	**0.0009**±0.0009

We have several key observations from Figure 6.11. Firstly, the proposed LIST model outperforms all baselines, which are consistent across all four dynamic networks. It demonstrates the advantages of leveraging network propagation in predicting link weights, and shows that the LIST model can capture the underlying structure of the network evolution. We also notice that the LIST model achieves a higher accuracy margin against the TMF model at early timestamps. It is because there is no enough data to train the TMF model, but the LIST model makes full use of the network structure through the propagation constraint.

Table 6.2 displays the average prediction RMSE across all timestamps of each dataset. It is

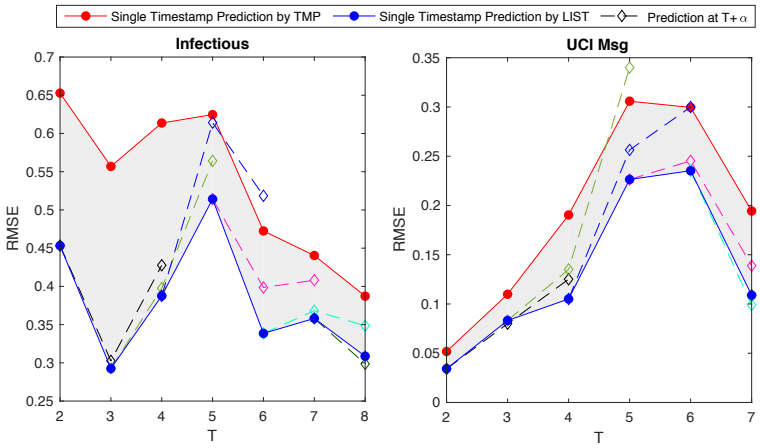


Figure 6.12: Multiple-timestamp prediction RMSE on *Infectious* and *UCI Msg* datasets

evident that the LIST model has a better average RMSE than all four competing methods. Note that the performance of the LIST model is about $18\times$ better than WCN on the *DBLP* dataset.

6.2.2.2 Multiple-timestamp Link Weight Prediction

Notably, the LIST model has the capability to effectively reconstruct the structure of the network at any given time. In this section, we run experiments to measure the prediction accuracy in predicting link weights for multiple timestamps. The task is to predict the link weights at timestamp $T + \alpha$, where $\alpha = \{1, 2, 3\}$ in the experiments. That is, we are using the LIST model to predict the next three timestamps. The rest of the parameters are set as Section 6.2.2.1. The multiple-timestamp prediction RMSE is presented in Figure 6.12.

Since we are predicting the link weight of the upcoming three timestamps, each short dash line has three dots (or fewer than three at the last two timestamps) which plot the RMSE of link prediction at timestamp $T + 1$, $T + 2$ and $T + 3$. There are $\mathcal{T} - 2$ dash lines in total for each dataset ($\mathcal{T}$ is the total number of timestamps). The dots on solid lines indicate the single-timestamp prediction RMSE of LIST and TMF, which are used as references. It can be seen that, even when $\alpha > 1$, the LIST prediction accuracy is still better than the baseline TMF most time. It is also worth mentioning that several predictions made by multiple-timestamp link prediction are better than single-timestamp predictions.

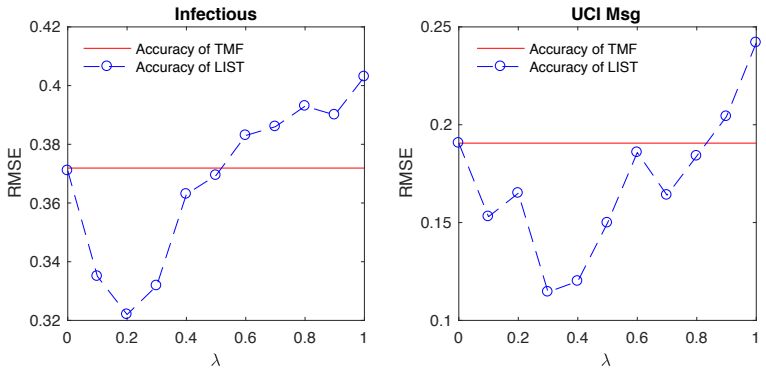


Figure 6.13: Prediction RMSE at timestamp T with different λ

6.2.2.3 Parameter Analysis

Propagation Balancing Parameter λ: The parameter λ defined by Eq. (6.16) specifies the relative amount of the information from its neighbors (propagation constraint) and its initial feature vectors (fitting constraint). By varying the parameter λ, we want to examine the relative importance of network structure in single-timestamp link weight prediction task. The special case of $\lambda = 0$ indicates that no network structure information will be considered. We choose different values of λ which varies from 0 to 1 with step size 0.1, and compute the prediction RMSE at timestamp T on *Infections* and *UCI Msg* datasets. All the remaining parameter settings are the same as Section 6.2.2.1. The results are summarized in Figure 6.13. The RMSE curves of both datasets follow a broad "U" shape. This means that as λ increases from 0 to 1, the prediction accuracy increases till an optimal value, after which it starts to decline. It is evident that the propagation constraint can help improve the performance of link prediction in dynamic networks. We also observe that $\lambda \in [0.1, 0.4]$ gives the optimal prediction RMSE.

Exponential Decay θ **and Factorization Rank** k: In this section, we conduct the parameter analysis on θ and k. θ regulates the exponential decay function $h(t)$. A larger θ represents that less weights are assigned to the previous timestamps. k is the latent dimension of the feature matrix $\mathbf{F}(t)$. We choose different values of θ varies from 0 to 1 with interval 0.1, and k varies from 20 to 100. We show the results on datasets *Infectious* and *UCI Msg*. The prediction RMSE is presented as a heat-map in Figure 6.14. It depicts that θ has a significant impact on prediction accuracy, and it is sensitive to the dataset. For the *Infectious* dataset, as θ increases, the proposed model has a

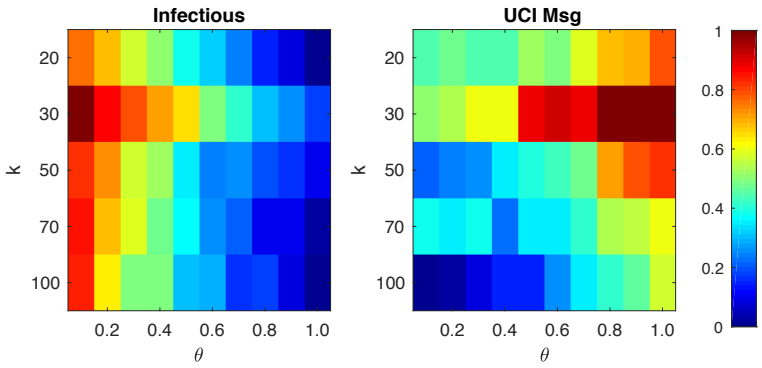


Figure 6.14: Prediction RMSE at timestamp T with different parameter pairs θ and k

better RMSE. This means a better prediction can be achieved when less weights are assigned to early timestamps. For the *UCI Msg* dataset, we observe a totally opposite trend. It also can be seen that RMSE improves with the increase of k. This is because larger k preserves more information when performs the matrix factorization. But from the complexity analysis in Section 6.2.1.3 we can see that a relatively small k takes less running time. Taking both running time and prediction accuracy into consideration, we choose a relative small k ranged from 20 to 100.

6.3 Summary

In this chapter, we first present NETWALK to detect anomalies in dynamic networks, by learning faithful network representations which can be updated dynamically as the network evolves over time. The latent network representations are learned using a number of network walks extracted from the initial network. The representations are obtained by clique embedding, which jointly minimizes the pairwise distance of vertex representations from each network walk, and the auto-encoder reconstruction error that serves as a global regularization. Based on the low-dimensional vertex representations, a clustering-based technique is employed to incrementally and dynamically detect network anomalies. Quantitative validation on anomaly detection task using four real-world datasets shows that NETWALK is computationally efficient and outperforms state-of-the-art techniques in anomaly detection. We then present a novel link prediction model, LIST, for dynamic networks which simultaneously incorporates network propagation and temporal matrix factorization techniques. This is guaranteed by the joint minimization of the network propagation

loss and the temporal network reconstruction error. The proposed model utilizes a user-defined sliding window to learn the parameters, thus supports streaming link prediction as well. Extensive experiments show that the LIST model outperforms the state-of-the-art techniques.

Part III

Conclusion and Future Directions

CHAPTER 7

Conclusion

The ubiquity and the emergence of complex networks provide computer science with a unique opportunity to build and design methods and applications upon. So, it is important to understand how such complex networks work, what is their structure, and how the structure of the networks influence the dynamical properties inside the networks. This thesis presents a combination of the methods and applications in static network analysis (PART I) and evolutionary network analysis (PART II). The research focus of this thesis is to analyze and model the structure, evolution and dynamics in the static and evolutionary information networks. Our contributions so far are the following,

On the analysis of *static network* side, we introduced the NETRA model to learn the network representations with adversarially regularized autoencoders. The resultant vertex representations can well capture the network structure through jointly considering both locality-preserving and global reconstruction constraints. The joint inference is encapsulated in a generative adversarial training process to circumvent the requirement of an explicit prior distribution, and thus obtains better generalization performance. We also introduced a novel scoring function called intensity score to measure the competitiveness of an advertiser, and proposed a community detection algorithm MAXINTENSITY to detect communities which have the maximum intensity score.

On the analysis of *evolutionary network* side, we introduced two novel time-dependent matrix factorization based models, TMF and COEVOL. These model have the advantage of significant generality in addressing various temporal applications because of its ability to explicitly represent the networks as a function of time. As specific examples, we provided results for temporal weight trend prediction, link prediction, dynamic community detection and event detection within the TMF framework, and we evaluated the extraordinary generality of COEVOL in terms of its appli-

cability to cross-network link prediction, lag correlation detection and community detection tasks. Even though we provided more general models, our results showed that their specific *instantiations* to different prediction tasks perform better than state-of-the-art techniques, thereby demonstrated the generality and effectiveness of the TMF and CoEvol frameworks. As for the applications in evolutionary network analysis, we proposed NetWalk for anomaly detection and LIST for link prediction. Compared with existing approaches, NetWalk has several advantages: 1) the network embedding can be updated dynamically, 2) streaming network nodes and edges can be encoded efficiently with constant memory space usage, 3). flexible to be applied on different types of networks, and 4) network anomalies can be detected in real-time. And the advantages LIST have are: 1) LIST uses a generic model to express the network structure as a function of time, which makes it also suitable for a wide variety of temporal network analysis problems beyond the focus of this paper; 2) by retaining the spatial and temporal consistency, LIST yields better prediction performance.

In the long run, we would like to model the evolution of networks both with attributes and label information, because a vast majority of real-world networks are coupled with a rich set of vertex attributes and labels, which could be potentially complementary in learning better network representations. Additionally, we also want to study the robustness of the network representation learning models against adversarial attacks. This is highly critical because adversaries are common and false data is easy to inject in the network-based learning applications. Next, we give a summary of contributions and the possible directions for future research.

7.1 Summary of Contributions

We summarize our contributions by grouping them into methods and applications under static and dynamic settings, as summarized in Table 7.1. In the *methods* column, we proposed NetRA to learn network representations under static setting, while TMF and CoEvol are proposed for evolutionary network analysis based on time-dependent matrix factorization. In the *Applications* column, we proposed MaxIntensity for community detection, NetWalk for anomaly detection and LIST for temporal link prediction.

Table 7.1: Structure of the thesis with references to the chapters

Thesis Parts	Methods	Applications
PART I: Static Network Analysis	Chapter 3	Chapter 4
	NETRA (KDD'18)	MAXINTENSITY (ICDM'15)
PART II: Evolutionary Network Analysis	Chapter 5	Chapter 6
	TMF (WSDM'17)	NETWALK (KDD'18)
	COEVOL (SDM'18)	LIST (IJCAI'17)

Methods:

1. We proposed a new deep network embedding model with adversarially regularized autoencoders, NETRA, to learn vertex representations by jointly minimizing locality-preserving loss and global reconstruction error using generative adversarial training process. The resultant representations are robust to the sparse inputs derived from the network. Experimental results demonstrate the effectiveness and efficiency of NETRA.

2. We developed TMF, a novel temporal matrix factorization model for dynamic network analysis. TMF has the advantage of significant generality in addressing various temporal applications because of its ability to explicitly represent the network as a function of time.

3. We proposed COEVOL to utilize a *shared temporal matrix factorization* framework to model co-evolution across multiple networks, which decomposes the adjacency matrix of each co-evolving network into a product of network-independent shared factor and a set of network-specific temporal factors, and impose a non-negativity constraint on the factors for greater interpretability.

Applications:

1. We proposed a new weighted bi-partite graph metric, *intensity score*, which captures the competition of the advertiser-keyword network better, and introduced a novel algorithm, MAXINTENSITY, based on *intensity score*, that detects advertiser communities with high competition.

2. We proposed NETWALK to detect anomalies in dynamic networks, by learning faithful network representations which can be updated dynamically as the network evolves over time. Quantitative validation on anomaly detection task showed that NETWALK is computationally efficient and outperforms state-of-the-art techniques.

3. We proposed a novel temporal link prediction model, LIST, for dynamic networks which simultaneously incorporates network propagation and temporal matrix factorization techniques. Extensive experiments show that the LIST model outperforms the state-of-the-art techniques.

7.2 Future Directions

Based on the recent results described in previous chapters and the research experience in information network analysis, we believe that the study of networks both with attributes and label information will be a promising and practical direction to develop information network analysis methods and applications, because the side information could be potentially complementary in learning better network representations. Additionally, we plan to study the robustness of the network representation learning models against adversarial attacks.

Attributed Network Embedding: A vast majority of real-world networks are coupled with a rich set of vertex attributes, which could be potentially complementary in learning better embeddings. Existing attributed network embedding models, with shallow or deep architectures, typically seek to match the representations in topology space and attribute space for each individual vertex by assuming that the samples from the two spaces are drawn uniformly. The assumption, however, can hardly be guaranteed in practice. Due to the intrinsic sparsity of sampled vertex sequences and the incomplete vertex attributes, the discrepancy between the attribute space and the network topology space inevitably exists. Furthermore, the interactions among vertex attributes, a.k.a cross features, have been largely ignored by existing approaches. It is important to develop an attributed network embedding model to address the above issues in order to achieve better generalization performance and robustness.

Dynamic Network Embedding: Network is dynamic and evolving over time in real world. Network analysis algorithms that ignore the dynamics of a given network can hardly capture sufficient information. The dynamic changes of networks are complex, including the addition and removal of vertices and edges, and the update of edge weights. It is challenging to design a framework for all these changes in an online fashion. A naive solution is to apply static embedding algorithms to each time-stamp of the evolutionary networks, which may face two issues: 1) latent representation space can hardly be aligned, 2) high computational cost to train the embedding models. We observe that a network may not change much in a short period of time, therefore the representation space should not change too much. A dynamic network embedding model is needed to efficiently update the representations against the small changes in the evolutionary networks.

Towards Robustness of Network Embedding Models: Deep learning models for graphs have achieved strong performance such as vertex classification and link prediction. Despite their proliferation, very few study of their robustness to adversarial attacks. Currently researches show that deep learning models for graphs can be easily fooled. Thus it is essential to develop a defense model to protect network embedding models from adversarial attack, especially in domains where network-based learning is used.

APPENDIX A

Appendix

A.1 Description of Datasets

In this section, we give brief descriptions and some of the basic statistics of the datasets used in this thesis from different domains, as shown in Table A.1.

- *Infectious* [Isella et al., 2011]: This network contains the daily dynamic contact networks collected during the Infectious SocioPatterns event that took place at the Science Gallery in Dublin, Ireland. Nodes represent exhibition visitors; edge weights represent face-to-face contact times.

- *UCI Message (UCI Msg)* [Opsahl and Panzarasa, 2009] is a directed communication network containing sent messages (edges) between the users (vertices) of an online community of students from the University of California Irvine.

- *arXiv hep-th* [Leskovec et al., 2007]: This collaboration network is from arXiv and covers scientific collaborations between authors and papers submitted to High Energy Physics - Theory category (hep-th). Vertices represent the authors, and edge weights between two authors represents the number of coauthored publications. Time-stamps denote the date of a publication.

- *Digg*[1]: This is the reply network of the news aggregator website *digg.com*. Each vertex is a Website user, and each weighted edge denotes the number of replies.

[1] `http://konect.uni-koblenz.de/networks`

Table A.1: Description of network datasets

Dataset	#Vertex	#Edge	Avg. Degree	Max. Weight	$\|T\|$	Type	#Label
Infectious	410	2,765	21.43	191	8 hours	Undirected	-
UCI Msg	1,899	13,838	14.57	98	7 months	Directed	-
arXiv hep-th	6,798	214,693	63.16	66	7 years	Undirected	-
Digg	30,360	85,155	5.61	25	14 days	Directed	-
Epinions	131,828	841,372	12.77	1	32 months	Directed	-
DBLP	315,159	743,709	4.72	159	34 years	Undirected	-
JDK	6,434	53,892	46.93	1	-	Directed	-
Blogcatalog	10,312	333,983	32.96	1	-	Undirected	-
PPI	3,890	76,584	19.69	1	-	Directed	50
Wikipedia	4,777	184,812	38.69	1	-	Directed	40

- *Epinions* [Massa and Avesani, 2006]: This is the trust and distrust network of Epinions, an online product rating site. The network contains individual users connected by directed trust and distrust links. Edges have the weight 1 for trust and -1 for distrust.

- *DBLP*[2] is an undirected collaboration graph of authors from the DBLP computer science bibliography. The vertices in this network represent the authors, and the edges represent the co-authorships between two authors.

- *JDK dependency (JDK)*[1] is the software class dependency network of the JDK 1.6.0.7 framework. The network is directed, with vertices representing Java classes and an edge between two vertices indicating there exists a dependency between the two classes.

- *Blogcatalog* [Tang and Liu, 2009] is an undirected social network from BlogCatalog website which manages the bloggers and their blogs. The vertices represent users and edges represent friendship between users.

- *Protein-Protein Interactions (PPI)* [Breitkreutz et al., 2007] is a subgraph of the PPI network

[2] `http://dblp.uni-trier.de/xml`

for Homo Sapiens, which is a network depicting interactions between human proteins. The vertex label indicates biological states of proteins.

- *Wikipedia* [Grover and Leskovec, 2016] is a directed word network. Vertex labels represent the Part-of-Speech (POS) tags inferred using the Stanford POS-Tagger [Toutanova et al., 2003].

A.2 Description of Baselines

This section summaries the baselines used in this thesis, including baselines for link prediction, community detection, anomaly detection and network embedding.

Link prediction baselines:

Let $\Gamma(x)$ denote the set of neighbors of vertex x in network $G(N, \boldsymbol{A}(t))$, and $w_{x,y}$ denote the link weights between nodes x and y.

- Common Neighbors (CN) [Liben-Nowell and Kleinberg, 2007]: For any pair of nodes x and y, the link prediction strategy is to define the $score(x, y) = |\Gamma(x) \cap \Gamma(y)|$, the number of common neighbors between x and y. The weighted version of common neighbors (WCN) [Murata and Moriyasu, 2007, Zhao et al., 2015] is defined as $score(x, y) = \sum_{z \in \Gamma(x) \cap \Gamma(y)} \frac{w_{x,z}+w_{y,z}}{2}$, but here we normalize the score by the size of common neighbors, thus $score(x, y) = \frac{1}{|\Gamma(x) \cap \Gamma(y)|} \sum_{z \in \Gamma(x) \cap \Gamma(y)} \frac{w_{x,z}+w_{y,z}}{2}$.

- Adamic Adar (AA) [Adamic and Adar, 2003]: This method assigns larger weights to less-connected neighbors. The measure is $score(x, y) = \sum_{z \in \Gamma(x) \cap \Gamma(y)} \frac{1}{log|\Gamma(z)|}$. The weighted version (WAA) [Zhao et al., 2015] is $score(x, y) = \sum_{z \in \Gamma(x) \cap \Gamma(y)} \frac{w_{x,z}+w_{y,z}}{2} \times \frac{1}{\log(1+s_z)}$, where $s_z = \sum_{z' \in \Gamma(z)} w_{z,z'}$.

- High-performance Link Prediction (HPLP) [Lichtenwalter et al., 2010]: This is a supervised classification framework to predict the new links. The weighted version is using the same features to train a regression model to predict the link weights.

- Preferential Attachment (PA) [Mitzenmacher, 2004]: This unweighted method corresponds to the measure $score(x, y) = |\Gamma(x)| \cdot |\Gamma(y)|$. The basic premise is that the probability that a new edge involves node x is proportional to $|\Gamma(x)|$.

- Nonparametric Link Prediction (NP) [Sarkar et al., 2012]: This method predicts links based on the features of its endpoints, as well as those of the local neighborhood around the endpoints.

- Link Prediction via Matrix Factorization (Fact-Sq) [Menon and Elkan, 2011]: This method solves the link prediction problem in graphs using a matrix factorization based approach. This baseline uses the square loss and the same latent dimension k as the proposed model in this thesis.

- CP Tensor Model (CP-Tensor) [Dunlavy et al., 2011]: This is a tensor-based method for predicting future links for bipartite graphs that evolve over time. In our problem setting, we apply this method to homogeneous graphs.

Community detection baselines:

- Information Maps (Infomap) [Rosvall and Bergstrom, 2007] is an information theoretic community detection approach that can be used with weighted and directed networks.

- Multilevel [Blondel et al., 2008] is a heuristic method based on modularity optimization that finds high modularity partitions of large networks in short time and discovers a hierarchical community structure for the network.

- Leading Eigenvector (Eigen) [Newman, 2006a] leverages a modularity matrix to maximize the modules in a network.

- Max Permanence [Chakraborty et al., 2014] detects the community structure by maximizing permanence score, a new vertex-based metric that can quantitatively give an estimate of the community-like structure of the network.

- Bi-partite Permanence (Bi-Permanence) is a modified version of Max-Permanence that detects the communities in a bipartite graph.

Anomaly detection baselines:

- GOutlier [Aggarwal et al., 2011] uses a structural connectivity model in order to define outliers in dynamic network. It designs a sampling method to maintain structural summaries of the underlying network.
- CM-Sketch [Ranshous et al., 2016] is an outlier detection model based on global and local structural properties of an edge stream. It utilizes Count-Min sketch for approximating these properties.

Network embedding baselines:

- Spectral Clustering (SC) [Tang and Liu, 2011] is an approach based on matrix factorization, generating the vertex representation with the smallest d eigenvectors of the normalized Laplacian matrix of the graph.
- DeepWalk [Perozzi et al., 2014b] is a skip-gram [Mikolov et al., 2013b] based model which learns the graph embedding with truncated random walks.
- node2vec [Grover and Leskovec, 2016] combines the advantage of breadth-first traversal and depth-first traversal algorithms. The random walks generated by node2vec can better represent the structural equivalence.
- Structural Deep Network Embedding (SDNE) [Wang et al., 2016] is a deep learning based network embedding model which uses autoencoder and locality-preserving constraint to learn vertex representations that capture the highly non-linear network structure.
- Adversarial Network Embedding (ANE) [Dai et al., 2017] proposes to train a discriminator to push the embedding distribution to match the fixed prior.

REFERENCES

[Adamic and Adar, 2003] Adamic, L. A. and Adar, E. (2003). Friends and neighbors on the web. *Social networks*, 25(3):211–230.

[Aggarwal and Subbian, 2014] Aggarwal, C. and Subbian, K. (2014). Evolutionary network analysis: A survey. *ACM Computing Surveys*, 47(1):10.

[Aggarwal, 2013] Aggarwal, C. C. (2013). *Outlier Analysis*. Springer.

[Aggarwal, 2016] Aggarwal, C. C. (2016). *Recommender Systems: The Textbook*. Springer.

[Aggarwal et al., 2010] Aggarwal, C. C., Zhao, Y., and Philip, S. Y. (2010). On clustering graph streams. In *SDM*, pages 478–489. SIAM.

[Aggarwal et al., 2011] Aggarwal, C. C., Zhao, Y., and Philip, S. Y. (2011). Outlier detection in graph streams. In *2011 IEEE 27th International Conference on Data Engineering*, pages 399–409. IEEE.

[Ailon et al., 2009] Ailon, N., Jaiswal, R., and Monteleoni, C. (2009). Streaming k-means approximation. In *NIPS*, pages 10–18.

[Akoglu and Faloutsos, 2010] Akoglu, L. and Faloutsos, C. (2010). Event detection in time series of mobile communication graphs. In *Army Science Conference*, pages 77–79.

[Akoglu and Faloutsos, 2013] Akoglu, L. and Faloutsos, C. (2013). Anomaly, event, and fraud detection in large network datasets. In *WSDM*, pages 773–774. ACM.

[Akoglu et al., 2010] Akoglu, L., McGlohon, M., and Faloutsos, C. (2010). Oddball: Spotting anomalies in weighted graphs. In *PAKDD*, pages 410–421. Springer.

[Akoglu et al., 2015] Akoglu, L., Tong, H., and Koutra, D. (2015). Graph based anomaly detection and description: a survey. *Data Mining and Knowledge Discovery*, 29(3):626–688.

[Al Hasan and Zaki, 2011] Al Hasan, M. and Zaki, M. J. (2011). A survey of link prediction in social networks. In *Social network data analytics*, pages 243–275. Springer.

[Arenas et al., 2006] Arenas, A., Díaz-Guilera, A., and Pérez-Vicente, C. J. (2006). Synchronization reveals topological scales in complex networks. *Physical review letters*, 96(11):114102.

[Arjovsky et al., 2017] Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein generative adversarial networks. In *ICML*, pages 214–223.

[Backstrom et al., 2006] Backstrom, L., Huttenlocher, D., Kleinberg, J., and Lan, X. (2006). Group formation in large social networks: membership, growth, and evolution. In *KDD*, pages 44–54. ACM.

[Bengio et al., 2003] Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *JMLR*, 3(Feb):1137–1155.

[Blondel et al., 2008] Blondel, V. D., Guillaume, J.-L., Lambiotte, R., and Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics*.

[Bojchevski and Günnemann, 2018] Bojchevski, A. and Günnemann, S. (2018). Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. In *ICLR*.

[Boone, 2008] Boone, J. (2008). A new way to measure competition. *The Economic Journal*, 118(531):1245–1261.

[Brandes et al., 2006] Brandes, U., Delling, D., Gaertler, M., Görke, R., Hoefer, M., Nikoloski, Z., and Wagner, D. (2006). *On modularity-np-completeness and beyond*. Citeseer.

[Breitkreutz et al., 2007] Breitkreutz, B.-J., Stark, C., Reguly, T., et al. (2007). The biogrid interaction database: 2008 update. *Nucleic acids research*, 36(suppl_1):D637–D640.

[Cao et al., 2015] Cao, S., Lu, W., and Xu, Q. (2015). Grarep: Learning graph representations with global structural information. In *CIKM*, pages 891–900. ACM.

[Cao et al., 2016] Cao, S., Lu, W., and Xu, Q. (2016). Deep neural networks for learning graph representations. In *AAAI*, pages 1145–1152.

[Chakraborty et al., 2014] Chakraborty, T., Srinivasan, S., Ganguly, N., Mukherjee, A., and Bhowmick, S. (2014). On the permanence of vertices in network communities. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1396–1405. ACM.

[Che et al., 2017] Che, T., Li, Y., Zhang, R., Hjelm, R. D., Li, W., Song, Y., and Bengio, Y. (2017). Maximum-likelihood augmented discrete generative adversarial networks. *arXiv preprint arXiv:1702.07983*.

[Cheng et al., 2016] Cheng, W., Zhang, K., Chen, H., Jiang, G., Chen, Z., and Wang, W. (2016). Ranking causal anomalies via temporal and dynamical analysis on vanishing correlations. In *KDD*, pages 805–814. ACM.

[Chi et al., 2007] Chi, Y., Song, X., Zhou, D., Hino, K., and Tseng, B. L. (2007). Evolutionary spectral clustering by incorporating temporal smoothness. In *KDD*, pages 153–162. ACM.

[Cormode and Muthukrishnan, 2005] Cormode, G. and Muthukrishnan, S. (2005). An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75.

[Dai et al., 2017] Dai, Q., Li, Q., Tang, J., and Wang, D. (2017). Adversarial network embedding. *arXiv preprint arXiv:1711.07838*.

[Defazio et al., 2014] Defazio, A., Bach, F. R., and Lacoste-Julien, S. (2014). Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *NIPS*, pages 1646–1654.

[Defferrard et al., 2016] Defferrard, M., Bresson, X., and Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, pages 3844–3852.

[Dong et al., 2017] Dong, Y., Chawla, N. V., and Swami, A. (2017). metapath2vec: Scalable representation learning for heterogeneous networks. In *KDD*, pages 135–144. ACM.

[Duan et al., 2014] Duan, L., Street, W. N., Liu, Y., and Lu, H. (2014). Community detection in graphs through correlation. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1376–1385. ACM.

[Dunlavy et al., 2011] Dunlavy, D. M., Kolda, T. G., and Acar, E. (2011). Temporal link prediction using matrix and tensor factorizations. *TKDE*, 5(2):10.

[Eberle and Holder, 2007] Eberle, W. and Holder, L. (2007). Anomaly detection in data represented as graphs. *Intelligent Data Analysis*, 11(6):663–689.

[Ellwardt et al., 2012] Ellwardt, L., Steglich, C., and Wittek, R. (2012). The co-evolution of gossip and friendship in workplace social networks. *Social Networks*, pages 623–633.

[Ermiş et al., 2015] Ermiş, B., Acar, E., and Cemgil, A. T. (2015). Link prediction in heterogeneous data via generalized coupled tensor factorization. *Data Mining and Knowledge Discovery*, 29(1):203–236.

[Fan et al., 2008] Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). Liblinear: A library for large linear classification. *JMLR*, 9(Aug):1871–1874.

[Farajtabar et al., 2015] Farajtabar, M., Wang, Y., Rodriguez, M., Li, S., Zha, H., and Song, L. (2015). Coevolve: A joint point process model for information diffusion and network coevolution. In *Advances in Neural Information Processing Systems*, pages 1945–1953.

[Fortunato, 2010] Fortunato, S. (2010). Community detection in graphs. *Physics Reports*, 486(3):75–174.

[Fortunato and Barthélemy, 2007] Fortunato, S. and Barthélemy, M. (2007). Resolution limit in community detection. *Proceedings of the National Academy of Sciences*, 104(1):36–41.

[Fu et al., 2009] Fu, W., Song, L., and Xing, E. P. (2009). Dynamic mixed membership blockmodel for evolving networks. In *ICML*, pages 329–336. ACM.

[Gao and Huang, 2018] Gao, H. and Huang, H. (2018). Deep attributed network embedding. In *IJCAI*, pages 3364–3370.

[Gao et al., 2010] Gao, J., Liang, F., Fan, W., Wang, C., Sun, Y., and Han, J. (2010). On community outliers and their efficient detection in information networks. In *SIGKDD*, pages 813–822. ACM.

[Gao et al., 2011] Gao, S., Denoyer, L., and Gallinari, P. (2011). Temporal link prediction by integrating content and structure information. In *CIKM*, pages 1169–1174. ACM.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. (2016). *Deep learning*, volume 1. MIT press Cambridge.

[Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *NIPS*, pages 2672–2680.

[Graepel et al., 2010] Graepel, T., Candela, J. Q., Borchert, T., and Herbrich, R. (2010). Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft's bing search engine. In *Proceedings of the 27th International Conference on Machine Learning*, pages 13–20.

[Grover and Leskovec, 2016] Grover, A. and Leskovec, J. (2016). node2vec: Scalable feature learning for networks.

[Gulrajani et al., 2017] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. (2017). Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*.

[Gupta et al., 2011] Gupta, M., Aggarwal, C. C., Han, J., and Sun, Y. (2011). Evolutionary clustering and analysis of bibliographic networks. In *ASONAM*, pages 63–70. IEEE.

[Gupta et al., 2014a] Gupta, M., Gao, J., Aggarwal, C. C., and Han, J. (2014a). Outlier detection for temporal data: A survey. *TKDE*, 9(26):2250–2267.

[Gupta et al., 2012a] Gupta, M., Gao, J., Sun, Y., and Han, J. (2012a). Community trend outlier detection using soft temporal pattern mining. In *ECML/PKDD*, pages 692–708. Springer.

[Gupta et al., 2012b] Gupta, M., Gao, J., Sun, Y., and Han, J. (2012b). Integrating community matching and outlier detection for mining evolutionary community outliers. In *SIGKDD*, pages 859–867. ACM.

[Gupta et al., 2014b] Gupta, M., Mallya, A., Roy, S., Cho, J. H., and Han, J. (2014b). Local learning for mining outlier subgraphs from network datasets. In *SDM*, pages 73–81.

[Hamilton et al., 2017] Hamilton, W. L., Ying, R., and Leskovec, J. (2017). Inductive representation learning on large graphs. *arXiv preprint arXiv:1706.02216*.

[Hastie et al., 2009] Hastie, T., Tibshirani, R., Friedman, J., Hastie, T., Friedman, J., and Tibshirani, R. (2009). *The elements of statistical learning*, volume 2. Springer.

[Hirschman, 1964] Hirschman, A. O. (1964). The paternity of an index. *The American Economic Review*, pages 761–762.

[Holder et al., 1994] Holder, L. B., Cook, D. J., Djoko, S., et al. (1994). Substucture discovery in the subdue system. In *KDD Workshop*, pages 169–180.

[Horvath, 1970] Horvath, J. (1970). Suggestion for a comprehensive measure of concentration. *Southern Economic Journal*, pages 446–452.

[Huang et al., 2017a] Huang, X., Li, J., and Hu, X. (2017a). Accelerated attributed network embedding. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pages 633–641. SIAM.

[Huang et al., 2017b] Huang, X., Li, J., and Hu, X. (2017b). Label informed attributed network embedding. In *WSDM*, pages 731–739. ACM.

[Hughes, 1996] Hughes, B. D. (1996). Random walks and random environments.

[Ide and Kashima, 2004] Ide, T. and Kashima, H. (2004). Eigenspace-based anomaly detection in computer systems. In *KDD*, pages 440–449. ACM.

[Indyk and Motwani, 1998] Indyk, P. and Motwani, R. (1998). Approximate nearest neighbors: towards removing the curse of dimensionality. In *ACM Symposium on Theory of Computing*, pages 604–613. ACM.

[Isella et al., 2011] Isella, L., Stehlé, J., Barrat, A., Cattuto, C., Pinton, J.-F., and Van den Broeck, W. (2011). What's in a crowd? analysis of face-to-face behavioral networks. *Journal of theoretical biology*, 271(1):166–180.

[Kashima et al., 2009] Kashima, H., Kato, T., Yamanishi, Y., Sugiyama, M., and Tsuda, K. (2009). Link propagation: A fast semi-supervised learning algorithm for link prediction. In *SDM*, pages 1100–1111. SIAM.

[Kim et al., 2017] Kim, Y., Zhang, K., Rush, A. M., LeCun, Y., et al. (2017). Adversarially regularized autoencoders for generating discrete structures. *arXiv preprint arXiv:1706.04223*.

[Kipf and Welling, 2016] Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

[Kloster and Gleich, 2014] Kloster, K. and Gleich, D. F. (2014). Heat kernel based community detection. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1386–1395. ACM.

[Kloumann and Kleinberg, 2014] Kloumann, I. M. and Kleinberg, J. M. (2014). Community membership identification from small seed sets. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1366–1375. ACM.

[Koren, 2010] Koren, Y. (2010). Collaborative filtering with temporal dynamics. *Communications of the ACM*, pages 89–97.

[Kuang et al., 2012] Kuang, D., Park, H., and Ding, C. H. (2012). Symmetric nonnegative matrix factorization for graph clustering. In *SDM*, volume 12, pages 106–117. SIAM.

[Lee and Seung, 1999] Lee, D. D. and Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791.

[Lee and Seung, 2001] Lee, D. D. and Seung, H. S. (2001). Algorithms for non-negative matrix factorization. In *NIPS*, pages 556–562.

[Leskovec et al., 2007] Leskovec, J., Kleinberg, J., and Faloutsos, C. (2007). Graph evolution: Densification and shrinking diameters. *TKDD*, 1(1):2.

[Levy and Goldberg, 2014] Levy, O. and Goldberg, Y. (2014). Neural word embedding as implicit matrix factorization. In *NIPS*, pages 2177–2185.

[Li et al., 2014] Li, X., Du, N., Li, H., Li, K., Gao, J., and Zhang, A. (2014). A deep learning approach to link prediction in dynamic networks. In *SDM*. SIAM.

[Liben-Nowell and Kleinberg, 2007] Liben-Nowell, D. and Kleinberg, J. (2007). The link-prediction problem for social networks. *JASIST*, 58(7):1019–1031.

[Lichtenwalter et al., 2010] Lichtenwalter, R. N., Lussier, J. T., and Chawla, N. V. (2010). New perspectives and methods in link prediction. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 243–252. ACM.

[Lü and Zhou, 2009] Lü, L. and Zhou, T. (2009). Role of weak ties in link prediction of complex networks. In *Proceedings of the 1st ACM international workshop on Complex networks meet information & knowledge management*, pages 55–58. ACM.

[Lü and Zhou, 2011] Lü, L. and Zhou, T. (2011). Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications*, 390(6):1150–1170.

[Maaten and Hinton, 2008] Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *JMLR*, 9(Nov):2579–2605.

[MacQueen et al., 1967] MacQueen, J. et al. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA.

[Makhzani et al., 2016] Makhzani, A., Shlens, J., Jaitly, N., and Goodfellow, I. (2016). Adversarial autoencoders. In *ICLR*.

[Mankad and Michailidis, 2013] Mankad, S. and Michailidis, G. (2013). Structural and functional discovery in dynamic networks with non-negative matrix factorization. *Physical Review E*, 88(4).

[Manzoor et al., 2016] Manzoor, E. A., Momeni, S., Venkatakrishnan, V. N., and Akoglu, L. (2016). Fast memory-efficient anomaly detection in streaming heterogeneous graphs. In *KDD*.

[Massa and Avesani, 2006] Massa, P. and Avesani, P. (2006). Trust-aware bootstrapping of recommender systems. In *ECAI workshop on recommender systems*, volume 28, page 29.

[McConville et al., 2015] McConville, R., Liu, W., and Miller, P. (2015). Vertex clustering of augmented graph streams. *SDM*.

[Menon and Elkan, 2011] Menon, A. K. and Elkan, C. (2011). Link prediction via matrix factorization. In *PKDD*, pages 437–452. Springer.

[Mikolov et al., 2013a] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

[Mikolov et al., 2013b] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119.

[Milgrom and Segal, 2002] Milgrom, P. and Segal, I. (2002). Envelope theorems for arbitrary choice sets. *Econometrica*, 70(2):583–601.

[Mitzenmacher, 2004] Mitzenmacher, M. (2004). A brief history of generative models for power law and lognormal distributions. *Internet mathematics*, 1(2):226–251.

[Moore, 1959] Moore, E. F. (1959). *The shortest path through a maze*. Bell Telephone System.

[Murata and Moriyasu, 2007] Murata, T. and Moriyasu, S. (2007). Link prediction of social networks based on weighted proximity measures. In *Web Intelligence, IEEE/WIC/ACM international conference on*, pages 85–88. IEEE.

[Newman, 2006a] Newman, M. E. (2006a). Finding community structure in networks using the eigenvectors of matrices. *Physical review E*, 74(3):036104.

[Newman, 2006b] Newman, M. E. (2006b). Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582.

[Ng, 2011] Ng, A. (2011). Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19.

[Noble and Cook, 2003] Noble, C. C. and Cook, D. J. (2003). Graph-based anomaly detection. In *SIGKDD*, pages 631–636. ACM.

[Opsahl and Panzarasa, 2009] Opsahl, T. and Panzarasa, P. (2009). Clustering in weighted networks. *Social networks*, pages 155–163.

[Oyama et al., 2011] Oyama, S., Hayashi, K., and Kashima, H. (2011). Cross-temporal link prediction. In *ICDM*, pages 1188–1193. IEEE.

[Perlich et al., 2012] Perlich, C., Dalessandro, B., Hook, R., Stitelman, O., Raeder, T., and Provost, F. (2012). Bid optimizing and inventory scoring in targeted online advertising. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 804–812. ACM.

[Perozzi and Akoglu, 2016] Perozzi, B. and Akoglu, L. (2016). Scalable anomaly ranking of attributed neighborhoods. In *SDM*, pages 207–215. SIAM.

[Perozzi et al., 2014a] Perozzi, B., Akoglu, L., Iglesias Sánchez, P., and Müller, E. (2014a). Focused clustering and outlier detection in large attributed graphs. In *SIGKDD*, pages 1346–1355.

[Perozzi et al., 2014b] Perozzi, B., Al-Rfou, R., and Skiena, S. (2014b). Deepwalk: Online learning of social representations. In *KDD*, pages 701–710. ACM.

[Radford et al., 2015] Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

[Rahman and Al Hasan, 2016] Rahman, M. and Al Hasan, M. (2016). Link prediction in dynamic networks using graphlet. In *PKDD*, pages 394–409. Springer.

[Rajeswar et al., 2017] Rajeswar, S., Subramanian, S., Dutil, F., Pal, C., and Courville, A. (2017). Adversarial generation of natural language. *arXiv preprint arXiv:1705.10929*.

[Ranshous et al., 2016] Ranshous, S., Harenberg, S., Sharma, K., Samatova, N. F., et al. (2016). A scalable approach for outlier detection in edge streams using sketch-based approximations. In *SDM*.

[Ranshous et al., 2015] Ranshous, S., Shen, S., Koutra, D., Harenberg, S., Faloutsos, C., and Samatova, N. F. (2015). Anomaly detection in dynamic networks: a survey. *Wiley Interdisciplinary Reviews: Computational Statistics*, 7(3):223–247.

[Ribeiro et al., 2017] Ribeiro, L. F., Saverese, P. H., and Figueiredo, D. R. (2017). Struc2vec: Learning node representations from structural identity. In *KDD*, pages 385–394. ACM.

[Rossi et al., 2013] Rossi, R. A., Gallagher, B., Neville, J., and Henderson, K. (2013). Modeling dynamic behavior in large evolving graphs. In *WSDM*, pages 667–676. ACM.

[Rosvall and Bergstrom, 2007] Rosvall, M. and Bergstrom, C. (2007). Maps of information flow reveal community structure in complex networks. In *In Proceedings of the National Academy of Sciences USA*.

[Roweis and Saul, 2000] Roweis, S. T. and Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326.

[Sarkar et al., 2012] Sarkar, P., Chakrabarti, D., and Jordan, M. I. (2012). Nonparametric link prediction in dynamic networks. In *ICML*, pages 1687–1694.

[Shapiro, 2010] Shapiro, C. (2010). The 2010 horizontal merger guidelines: From hedgehog to fox in forty years. *Antitrust Law Journal*, pages 49–107.

[Sun et al., 2007] Sun, J., Faloutsos, C., Papadimitriou, S., and Yu, P. S. (2007). Graphscope: parameter-free mining of large time-evolving graphs. In *KDD*, pages 687–696. ACM.

[Sun et al., 2005] Sun, J., Qu, H., Chakrabarti, D., and Faloutsos, C. (2005). Neighborhood formation and anomaly detection in bipartite graphs. In *ICDM*, pages 8–17. IEEE.

[Sun et al., 2006] Sun, J., Tao, D., and Faloutsos, C. (2006). Beyond streams and graphs: dynamic tensor analysis. In *KDD*, pages 374–383. ACM.

[Sun et al., 2014] Sun, Y., Tang, J., Han, J., Chen, C., and Gupta, M. (2014). Co-evolution of multi-typed objects in dynamic star networks. *TKDE*, 26(12):2942–2955.

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112.

[Tang et al., 2015] Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., and Mei, Q. (2015). Line: Large-scale information network embedding. In *WWW*, pages 1067–1077. ACM.

[Tang and Liu, 2009] Tang, L. and Liu, H. (2009). Relational learning via latent social dimensions. In *KDD*, pages 817–826. ACM.

[Tang and Liu, 2011] Tang, L. and Liu, H. (2011). Leveraging social media networks for classification. *Data Mining and Knowledge Discovery*, 23(3):447–478.

[Tang et al., 2008] Tang, L., Liu, H., Zhang, J., and Nazeri, Z. (2008). Community evolution in dynamic multi-mode networks. In *KDD*, pages 677–685. ACM.

[Theocharidis et al., 2009] Theocharidis, A., Van Dongen, S., Enright, A. J., and Freeman, T. C. (2009). Network visualization and analysis of gene expression data using biolayout express3d. *Nature protocols*, 4(10):1535–1550.

[Tian et al., 2014] Tian, F., Gao, B., Cui, Q., Chen, E., and Liu, T.-Y. (2014). Learning deep representations for graph clustering. In *AAAI*, pages 1293–1299.

[Toutanova et al., 2003] Toutanova, K., Klein, D., Manning, C. D., and Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. pages 173–180. Association for Computational Linguistics.

[Tylenda et al., 2009] Tylenda, T., Angelova, R., and Bedathur, S. (2009). Towards time-aware link prediction in evolving social networks. In *Proceedings of the 3rd workshop on social network mining and analysis*, page 9. ACM.

[Vasiloglou et al., 2009] Vasiloglou, N., Gray, A. G., and Anderson, D. V. (2009). Non-negative matrix factorization, convexity and isometry. In *SDM*, pages 673–684. SIAM.

[Villani, 2008] Villani, C. (2008). *Optimal transport: old and new*, volume 338. Springer Science & Business Media.

[Wang et al., 2007] Wang, C., Satuluri, V., and Parthasarathy, S. (2007). Local probabilistic models for link prediction. In *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pages 322–331. IEEE.

[Wang et al., 2016] Wang, D., Cui, P., and Zhu, W. (2016). Structural deep network embedding. In *SIGKDD*, pages 1225–1234. ACM.

[Wang et al., 2018] Wang, H., Wang, J., Wang, J., Zhao, M., Zhang, W., Zhang, F., Xie, X., and Guo, M. (2018). Graphgan: Graph representation learning with generative adversarial nets. *AAAI*.

[Watts and Strogatz, 1998] Watts, D. J. and Strogatz, S. H. (1998). Collective dynamics of 'small-world'networks. *nature*, 393(6684):440–442.

[Weston et al., 2008] Weston, J., Ratle, F., and Collobert, R. (2008). Deep learning via semi-supervised embedding. In *ICML*.

[Wu, 1982] Wu, F.-Y. (1982). The potts model. *Reviews of modern physics*, 54(1):235.

[Yang et al., 2015] Yang, C., Liu, Z., Zhao, D., Sun, M., and Chang, E. Y. (2015). Network representation learning with rich text information. In *IJCAI*, pages 2111–2117.

[Yu et al., 2017a] Yu, L., Zhang, W., Wang, J., and Yu, Y. (2017a). Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI*, pages 2852–2858.

[Yu et al., 2013a] Yu, W., Aggarwal, C. C., Ma, S., and Wang, H. (2013a). On anomalous hotspot discovery in graph streams. In *ICDM*, pages 1271–1276. IEEE.

[Yu et al., 2017b] Yu, W., Aggarwal, C. C., and Wang, W. (2017b). Temporally factorized network modeling for evolutionary network analysis. In *WSDM*, pages 455–464. ACM.

[Yu et al., 2017c] Yu, W., Cheng, W., Aggarwal, C. C., Chen, H., and Wang, W. (2017c). Link prediction with spatial and temporal consistency in dynamic networks. In *IJCAI*, pages 3343–3349. AAAI Press.

[Yu et al., 2013b] Yu, W., Zeng, G., Luo, P., Zhuang, F., He, Q., and Shi, Z. (2013b). Embedding with autoencoder regularization. In *ECML/PKDD*, pages 208–223. Springer.

[Yuster and Zwick, 2005] Yuster, R. and Zwick, U. (2005). Fast sparse matrix multiplication. *ACM Transactions on Algorithms*, 1(1):2–13.

[Zachary, 1977] Zachary, W. W. (1977). An information flow model for conflict and fission in small groups. *Journal of anthropological research*, 33(4):452–473.

[Zhang et al., 2016] Zhang, D., Yin, J., Zhu, X., and Zhang, C. (2016). Homophily, structure, and content augmented network representation learning. In *ICDM*, pages 609–618. IEEE.

[Zhao et al., 2015] Zhao, J., Miao, L., Yang, J., Fang, H., Zhang, Q.-M., Nie, M., Holme, P., and Zhou, T. (2015). Prediction of links and weights in networks by reliable routes. *Scientific reports*, 5.

[Zhao and Yu, 2013] Zhao, Y. and Yu, P. (2013). On graph stream clustering with side information. In *SDM*, pages 139–150. SIAM.

[Zheleva et al., 2009] Zheleva, E., Sharara, H., and Getoor, L. (2009). Co-evolution of social and affiliation networks. In *KDD*, pages 1007–1016. ACM.

[Zhou et al., 2017] Zhou, C., Liu, Y., Liu, X., Liu, Z., and Gao, J. (2017). Scalable graph embedding for asymmetric proximity. In *AAAI*, pages 2942–2948.

[Zhou et al., 2003] Zhou, D., Bousquet, O., Lal, T. N., Weston, J., and Schölkopf, B. (2003). Learning with local and global consistency. In *NIPS*, volume 16, pages 321–328.

[Zhou and Lipowsky, 2004] Zhou, H. and Lipowsky, R. (2004). Network brownian motion: A new method to measure vertex-vertex proximity and to identify communities and subcommunities. In *Computational Science-ICCS 2004*, pages 1062–1069. Springer.

[Zhu et al., 2016] Zhu, L., Guo, D., Yin, J., Ver Steeg, G., and Galstyan, A. (2016). Scalable temporal latent space inference for link prediction in dynamic social networks. *TKDE*, 28(10):2765–2777.