Network Embedding and Its Applications

from Static Networks to Dynamic Networks

by

Chengbin Hou

A thesis submitted to
The University of Birmingham
for the degree of
Doctor of Philosophy

School of Computer Science
College of Engineering and Physical Sciences
The University of Birmingham
June 2021

ABSTRACT

Apart from the attached attributes of entities, the relationships among entities are also an important perspective that reveals the topological structure of entities in a complex system. A network (or graph) with nodes representing entities and links indicating relationships, has been widely used in sociology, biology, chemistry, medicine, the Internet, etc. However, traditional machine learning and data mining algorithms, designed for the entities with attributes (i.e., data points in a vector space), cannot effectively and/or efficiently utilize the topological information of a network formed by relationships among entities. To fill this gap, Network Embedding (NE) is proposed to embed a network into a low dimensional vector space while preserving some topologies and/or properties, so that the resulting embeddings can facilitate various downstream machine learning and data mining tasks.

Although there have been many successful NE methods, most of them are designed for embedding static plain networks. In fact, real-world networks often come with one or more additional properties such as node attributes and dynamic changes. The central research question of this thesis is "where and how can we apply NE for more realistic scenarios?". To this end, we propose three novel NE methods, each of which is for addressing the new challenges resulting from one type of more realistic networks. Besides, we also discuss the applications of NE with the focus to the drug-target interaction prediction problem.

To be more specific, *first*, we investigate how to embed the attributed network, which can better describe a real-world complex system by including node attributes to a network. Previous Attributed Network Embedding (ANE) methods cannot effectively embed attributed networks especially when networks become sparse, and/or are not scalable to large-scale networks. To deal with these challenges, we propose a scalable ANE method to effectively and robustly embed attributed networks with different sparsities. *Second*, we study how to embed the dynamic network, which is often the case in real-world scenarios as real-world complex systems often evolve over time. Most previous Dynamic Network Embedding (DNE) methods try to capture the topological changes at or around the most affected nodes and accordingly update node embeddings. Unfortunately, this kind of approximation, although

can improve efficiency, cannot effectively preserve the global topology of a dynamic network at each timestep, due to not considering the inactive sub-networks that receive accumulated topological changes propagated via the high-order proximity. To tackle this challenge, we propose a DNE method for better global topology preservation. *Third*, comparing to static networks, dynamic networks have a unique character called the degree of changes, which can be used to describe a kind of dynamic character of an input dynamic network about its rate of streaming edges between consecutive snapshots. The degree of changes could be very different for different dynamic networks. However, it remains unknown if existing DNE methods can robustly obtain good effectiveness to different degrees of changes, in particular for corresponding dynamic networks generated from the same dataset by different slicing settings. To answer this open question, we test several state-of-the-art DNE methods, and then further propose a DNE method that can more robustly obtain good effectiveness to the dynamic networks with different degree of changes. *Fourth*, regarding a specific application of NE to a real-world problem, we propose a NE based Drug-Target Interaction (DTI) prediction method by additionally utilizing the two implicit networks which are extracted from a given DTI network but are ignored in previous DTI prediction methods. A case study indicates that the proposed method can predict novel DTIs.

Key Words: *Network Embedding; Dynamic Network Embedding; Attributed Networks; Dynamic Networks; Robustness; Representation Learning; Machine Learning; Data Mining; Drug Discovery*

ACKNOWLEDGMENTS

As a student of the Joint PhD Training Program offered by the University of Birmingham (UoB) and Southern University of Science and Technology (SUSTech), I would like to thank Dr. Shan He in UoB and Prof. Ke Tang in SUSTech for their supervision, as well as UoB and SUSTech for their support for my studies in both universities. Particularly, I would like to express my heartfelt gratitude to my supervisors for their professional advice, encouragement, and kindness. They were always willing to answer my questions and provided useful feedback for me. I learned a lot from them, especially about critical thinking and the way to conduct research. I would not complete this thesis without their guidance.

I am sincerely grateful to several people, mostly from UoB and SUSTech, Prof. Iain Style, Prof. Xin Yao, Prof. Ata Kaban, Dr. Yunwen Lei, Dr. Peng Yang, Dr. Bo Yuan, Dr. Xiaofeng Lu, Dr. Wenjing Hong, Dr. Guiyin Li, Dr. Shencai Liu, Ms. Sarah Brookes and Ms. Kate Sterne (UoB student administrators), Mr. Xuefeng Zhang and Mr. Xing Zhang (SUSTech student administrators), Dr. Zheng Hu (from Huawei), and Dr. Bingzhe Wu (from Tencent) for their insightful discussions in my research or their helps from other aspects.

My special thanks go to my buddies in Birmingham and Shenzhen, Yu Zhang, Guoji Fu, Han Zhang, Phan Trung Hai Nguyen, Fuad Mire Hassan, Robeter Chin, David McDonald, Gourab Ghosh Roy, Abdullabh Alharbi, Irfan Muhammad, Shaolong Shi, Jinbao Wang, Hua Yan, Jikai Wu, Boping Deng, Jinxin Sun, Rui He, Muyao Zhong, Qi Yang, Zhiyuan Wang, and many others, for countless enjoyable experiences with you during my Ph.D. journey.

I would also like to thank Prof. Peter Tino and Prof. Yannis Goulermas for agreeing to be the examiners of this thesis and my viva.

Finally, this thesis is dedicated to my parents, my wife, and my daughter for their love, understanding, and encouragement. They put me where I am today.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Network Embedding (NE), also known as graph embedding, network representation learning, or graph representation learning, has attracted considerable attention especially from data mining, machine learning, and network science communities, due to its widespread real-world applications of relational data in sociology, biology, chemistry, medicine, the Internet, etc. In this chapter, the general background and motivation of NE are presented first. After that, we summarize the research questions and contributions of this thesis. The outline of the thesis is given at the end of this chapter.

1.1 Background

Nowadays, a huge amount of web data and sensor data are generated every day and everywhere [1]. According to the report by Demo Inc. [2], for *every minute* in 2020, Facebook users share 150,000 messages; Amazon ships 6659 packages; Venmo users send $ 239,196 worth of payments; Zoom hosts 208,333 participants in meetings; 1,388,889 people make video or voice calls; ect. The so-called big data enables researchers from various communities such as data mining, machine learning, and network science, to develop and verify their algorithms towards the large-scale real-world applications.

The real-world entities can be abstracted as data points, and the attached attributes of

each entity can be preprocessed as a vector. For example, one may treat social users as data points as shown in Figure 1.1 a), and preprocess their activities [3] or profiles [4] into vectors. For a friend recommendation system, one straightforward idea is to adopt those vectors as the input of the friend recommendation system for further computing.

In fact, the inherent relationships between entities exist in many real-world scenarios such as messages exchanged between users in Facebook, packages shipped between addresses in Amazon, and payments made between users in Venmo. Following the aforementioned example, except the attached attributes of users, they are also linked if two users have a friendship as shown in Figure 1.1 b), which naturally forms a network.

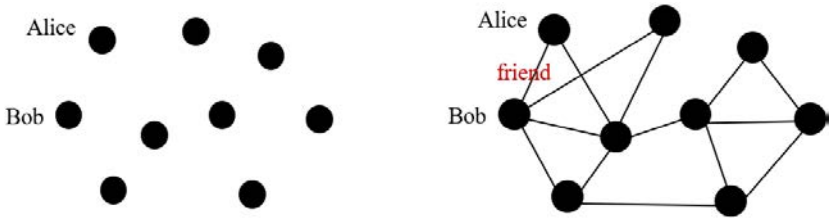


Figure 1.1: Two kinds of data: a) data points, and b) networked data points.

A *network* (or graph), with nodes (or vertexes) representing entities and links (or edges) indicating relationships, has been widely used in sociology, biology, chemistry, medicine, the Internet, etc. Just take a few examples as shown in Figure 1.2.

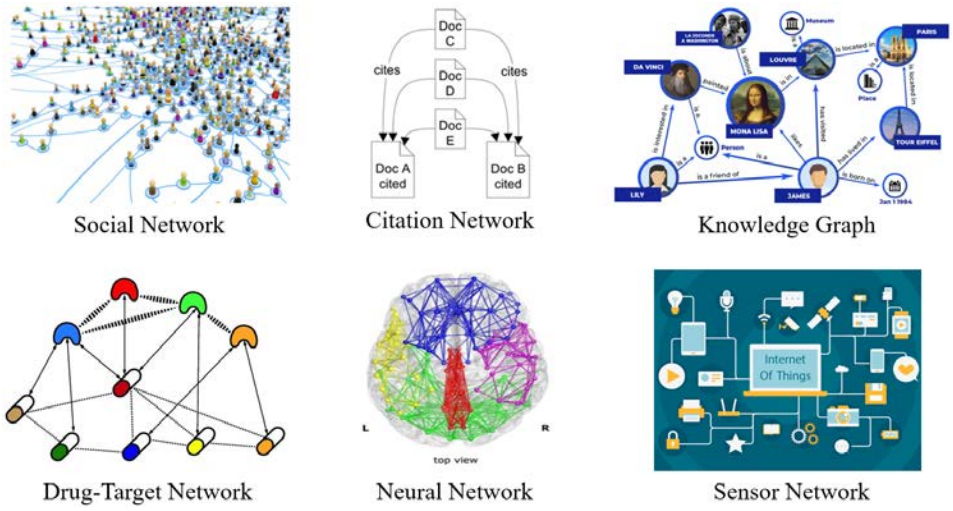


Figure 1.2: A few examples of real-world networks. Subfigures are adapted from Internet.

The relationship between two entities in a complex system can be easily modeled as a link connecting two nodes in a network. In this way, a complex system yields a new perspective about its network topology. The *network topology* is important information to study the relationships among (may be more than two, e.g., neighbors of neighbors) entities, and even investigate the characteristics of a whole system. In fact, the network topological information has been explicitly or implicitly exploited in the literature.

- Previous studies in sociology and biology [5–7] have shown that the topological information and attribute information of entities are highly correlated with each other (so-called homophily). Using both information for network inference tasks may achieve better results as two sources of information can help each other.
- In manifold learning, one popular algorithm is Local Linear Embeddings (LLE) [8]. It first constructs the local neighborhood graph using only k-nearest neighbors for each data point. The graph thus reveals the invariant geometric properties of each data point to its k-nearest neighbors [9]. LLE assumes data points are sampled from d-dimensional manifold. As a result, it then learns d-dimensional data embeddings while preserving the invariant geometric properties given by the weights in the graph.
- The Convolutional Neural Networks (CNN) has received a great success in computer vision related problems [10]. An input image can be viewed as a special network, where each pixel (except the pixels at boundary) is linked with its eight adjacent pixels. Each pixel is regarded as a node and its neighbors are defined by the filter size, e.g., 3 by 3, so that the 2D sliding window takes weighted average of pixel values of the node along with its neighbors [11]. It might be worth noting that CNN can now be interpreted as an operation over the special network formed by adjacent pixels.

To utilize network topological information, one common idea might be directly using the *adjacency matrix* of a network. Specifically, the vector in the i-th row stores the relationships between the i-th node to all nodes in a network, which can be treated as the topological feature of the i-th node. The vector carrying the topological feature of the i-th node is then fed to a downstream machine learning or data mining task. Unfortunately, the such

straightforward vector representation would face the following challenges.

- The i-th row vector of an adjacency matrix only reflects the first order proximity (i.e., the direct neighboring information) between i-th node to all nodes in a network. However, the second order proximity (i.e., neighbors of neighbors) and the higher order proximity, which might also be useful knowledge for a downstream predictive task, are not directly included in the row vector of an adjacency matrix.
- Many real-world networks are larger-scale, e.g., a social network may have millions of users. If a network has 10^6 nodes, the size of its adjacency matrix becomes 10^6 by 10^6. The row vector carrying the topological information of a node then has 10^6 dimensions. Feeding vectors with such high dimensionality to a downstream machine learning or data mining algorithm is likely to lead to a famous phenomenon called the curse of dimensionality [12], which would reduce the performance of downstream tasks.
- Due to the time-evolving nature of real-world complex systems, most real-world networks are dynamic by nature. Under the dynamic environment, when new nodes add to the network, the dimensionality of vectors would vary if we directly take out the vector from the adjacency matrix. The varying dimensionality of vectors becomes problematic while feeding them to a downstream machine learning or data mining task.

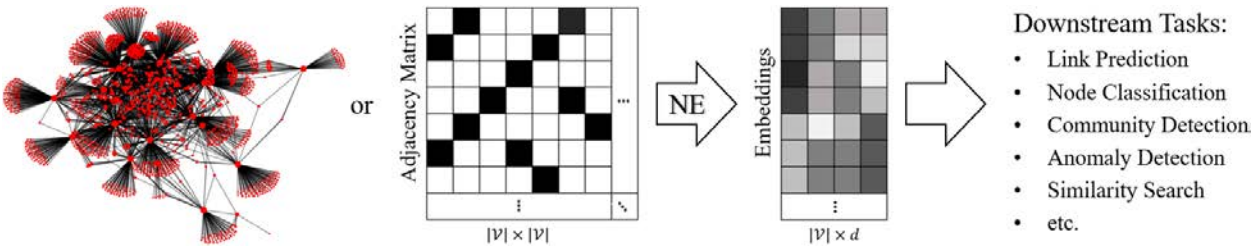


Figure 1.3: The workflow of Network Embedding (NE).

To address these challenges, *Network Embedding* (NE) is proposed to embed a network into a low dimensional continuous vector space (where the specified embedding dimension d is often much less than the number of nodes $|\mathcal{V}|$ for a large-scale network, i.e., $d \ll |\mathcal{V}|$) while preserving some network topologies and/or properties, so that the resulting embeddings can

facilitate various downstream machine learning and data mining tasks, e.g., link prediction and node classification. The workflow of NE is illustrated as shown in Figure 1.3.

1.2 Research Questions

Although there have been many successful NE methods, most of them are designed for embedding static plain networks (Figure 1.4-a). In fact, real-world networks often come with one or more additional properties such as node attributes (Figure 1.4-b) and dynamic changes (Figure 1.4-c). The central research question of this thesis is *"where and how can we apply NE for more realistic scenarios?"*. To this end, we propose three novel NE methods, each of which is for addressing the new challenges resulting from one type of more realistic networks. Besides, we also discuss the applications of NE to the drug-target interaction prediction problem (Figure 1.4-d), a key step to drug discovery [13].

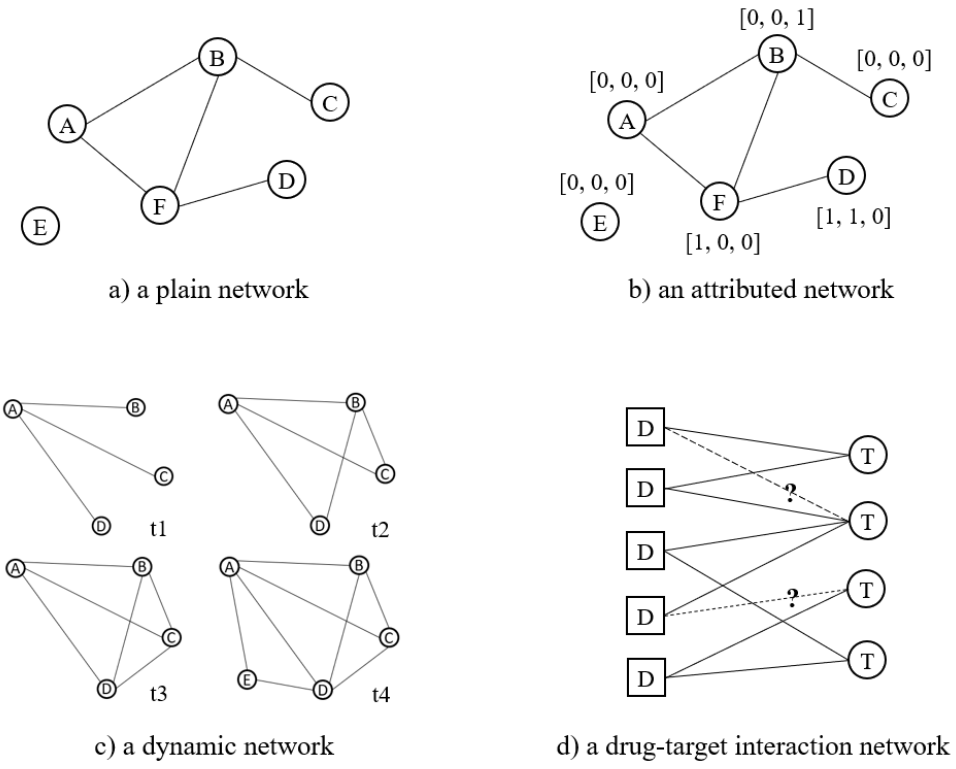


Figure 1.4: Different types of networks this thesis is interested in: plain networks, attributed networks, dynamic networks, and drug-target interaction networks (bipartite networks).

1.2.1 How to embed an attributed sparse network

Comparing to a plain network, an attributed network has better capability to describe a complex system, since it not only reflects the topological information of the system but also records the attribute information of entities in the system. Attributed Network Embedding (ANE), which aims to learn enhanced node embeddings by considering both topological and attribute information, is attracting much attention [14–19].

However, the existing ANE methods still face some challenges. First, most existing ANE works have not carefully considered the issues caused by sparse networks, and accordingly proposed a special treatment to explicitly tackle the issues. As a result, they may obtain poor node embeddings on sparse or extremely sparse networks, especially for those methods [15, 17, 19–22] relying on edges to process node attributes. Note that, the sparse network is a common real-world scenario [23]. Second, the scalability of ANE problem becomes a more challenging problem compared with plain network embedding problem, as the methods to solve ANE problem require additional computational resources to also utilize node attributes. And third, it is worth noticing the robustness of ANE methods to different networks or the same network with different sparsities regarding parameter tuning. Instead of grid search for the best hyper-parameter(s) [14, 16, 17, 20, 21], the robustness is desirable to design a generic ANE method. To be more specific, it should have relatively good and stable performance for few typical choices or by following explicit tuning advice [24], so that others can easily apply it in different real-world applications without much effort on hyper-parameter tuning, especially if it already has several other hyper-parameters to tune [25].

Research Questions 1 (RQ1): How can we embed an attributed sparse network effectively, efficiently, and robustly? Specifically, how do we use attribute information to alleviate the issue caused by topological sparse networks? What is and how do we handle the bottleneck that would restrict the scalability of the proposed ANE method? Does the proposed ANE method have a (or a few) preferred hyper-parameter setting(s) which can obtain a satisfactory performance robustly to different networks or the same network with different sparsities?

1.2.2 How to embed a dynamic network with global topology preservation

Many real-world networks are dynamic by nature, i.e., edges might be added or deleted between seen and/or unseen nodes as time goes on. For instance, in a wireless sensor network, devices will regularly connect to or accidentally disconnect from routers; in a social network, new friendships will establish between new users and/or existing users. Due to the time-evolving nature of many real-world networks, Dynamic Network Embedding (DNE) is now attracting much attention [26–36]. The main and common objective of DNE is to efficiently update node embeddings while preserving network topology at each time step. Most existing DNE methods try to compromise between effectiveness (evaluated by downstream tasks) and efficiency (while obtaining node embeddings). The idea is to capture the topological changes at or around the most affected nodes (instead of all nodes), and promptly update node embeddings based on an efficient incremental learning paradigm.

Unfortunately, this kind of approximation, although can improve the efficiency, cannot effectively preserve the global topology of a dynamic network at each time step. Concretely, any changes, i.e., edges being added or deleted, would affect all nodes in a connected network and greatly modify the proximity between nodes over a network via the high-order proximity. On the other hand, the real-world dynamic networks usually have some inactive sub-networks where no change occurs lasting for several time steps. Putting both together, the existing DNE methods that focus on the most affected nodes (belonging to the active sub-networks) but do not consider the inactive sub-networks, would overlook the accumulated topological changes propagating to the inactive sub-networks via the high-order proximity.

Research Questions 2 (RQ2): How can we embed a dynamic network with global topology preservation? Specifically, why do we need global topology preservation for DNE? What is the strategy to achieve the global topology preservation? Does the proposed DNE method with global topology preservation indeed improve the performance compared to that without global topology preservation?

1.2.3 How to embed a dynamic network robustly to degree of changes

In the context of dynamic environment, the robustness of a method itself is an important problem, as the dynamic environment would bring in many uncertainties comparing to the static environment. For a DNE problem, generating an input dynamic network, which consists of a series of snapshots, also involves many uncertainties. Regarding an input dynamic network, the degree of changes is a unique character of dynamic networks comparing to static networks. It can be used to describe a kind of dynamic character of an input dynamic network about its rate of streaming edges between consecutive snapshots. It could be very different in real-world scenarios, e.g., the degree of changes would be different for a daily updated network and a weekly updated network. Moreover, different slicing settings over the same dataset such as increasing by one hundred edges per snapshot and one thousand edges per snapshot, would also lead to dynamic networks with different degrees of changes. Unfortunately, existing DNE works [26–29, 31–35, 37–47] have not considered the effect of different degree of changes of an input dynamic network to DNE methods. As a result, these methods might not be robust enough to different degree of changes even if the corresponding input dynamic networks come from the same dataset. However, the robustness of DNE to different degree of changes is a desirable characteristic, as this would improve the reliability and usability of the DNE method while applying it to unknown real-world applications.

Research Questions 3 (RQ3): How can we embed a dynamic network robustly to the degree of changes? Specifically, why do we investigate the robustness of DNE w.r.t. the degree of changes? Are the existing DNE methods robust enough w.r.t. the degree of changes? Can we propose a more robust DNE method?

1.2.4 How to apply network embedding to drug-target interaction prediction

The reason why Network Embedding (NE) becomes such popular nowadays could be mainly owing to its widespread real-world applications. NE has been not only employed to some

generic applications (or called downstream tasks) such as link prediction and node classification [48–53] as shown in Figure 1.3, but also applied to some specific applications such as clustering vehicle trajectory in transportation systems [54, 55] and predicting unknown drug-target interaction in drug discovery [56, 57].

Research Questions 4 (RQ4): How can we apply NE to Drug-Target Interaction (DTI) prediction? Specifically, how do we reformulate DTI prediction problem as a link prediction problem? What networks should we construct based on a DTI dataset? Does the newly proposed NE based DTI prediction method really work? Beyond benchmarking, can we interpret the DTI prediction outcomes in terms of drug discovery via a case study?

1.3 Contributions of the Thesis

The main contributions of this thesis can be briefly summarized as follows:

- discover and suggest three sets of new challenges resulting from more realistic networks, one of which is from attributed networks and two of which are from dynamic networks (Chapter 3, 4, and 5);
- propose and empirically study three novel NE methods, each of which is for addressing one set of the new challenges (Chapter 3, 4, and 5);
- propose and empirically study a novel NE based drug-target interaction prediction method, which applies NE to extract additional features from the implicit networks constructed from an input drug-target interaction network (Chapter 6);
- provide sufficient reading materials of NE such as motivation, literature, concepts, representative approaches, and future directions (Chapter 1, 2, and 7).

The specific contributions for Chapter 3, 4, 5, and 6 are summarized and clearly presented in each chapter respectively. Therefore, it might not be necessary to repeat here.

1.4 Publications Resulting from the Thesis

Work 1: **Chengbin Hou**, Shan He, and Ke Tang. RoSANE: Robust and Scalable Attributed Network Embedding for Sparse Networks. *Neurocomputing*, 2020. [58]

▶ Chapter 3 mainly presents Work 1.

Work 2: **Chengbin Hou**, Han Zhang, Shan He, and Ke Tang. GloDyNE: Global Topology Preservation Dynmaic Network Embedding. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2020. [59]

▶ Chapter 4 mainly presents Work 2. See also the invited extended abstract [60].

Work 3: **Chengbin Hou**, Guoji Fu, Peng Yang, Zheng Hu, Shan He, and Ke Tang. Robust Dynamic Network Embedding via Ensembles. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, submitted. [61]

▶ Chapter 5 mainly presents Work 3.

Work 4: Han Zhang, **Chengbin Hou**, David McDonald, and Shan He. A Network Embedding Based Approach to Drug-Target Interaction Prediction Using Additional Implicit Networks. *International Conference on Artificial Neural Networks (ICANN)*, 2021. [62]

▶ Chapter 6 mainly presents Work 4.

Work 5: **Chengbin Hou** and Ke Tang. Towards Robust Dynamic Network Embedding. *International Joint Conference on Artificial Intelligence (IJCAI)*, DC track, 2021. [63]

▷ Chapter 7 mentions Work 5.

Work 6: Muyao Zhong, **Chengbin Hou**, and Ke Tang. A Neural Embedding Model for Drug-Target Interaction Prediction with Micro and Macro Proximities Preservation. in preparation.

▷ Chapter 7 mentions Work 6.

Work 7: Guoji Fu, **Chengbin Hou**, and Xin Yao. Learning Topological Representation for Networks via Hierarchical Sampling. *International Joint Conference on Neural Networks (IJCNN)*, 2019. [64]

▷ Chapter 3 mentions Work 7.

1.5 Outline of the Thesis

The outline of this thesis is illustrated in Figure 1.5.

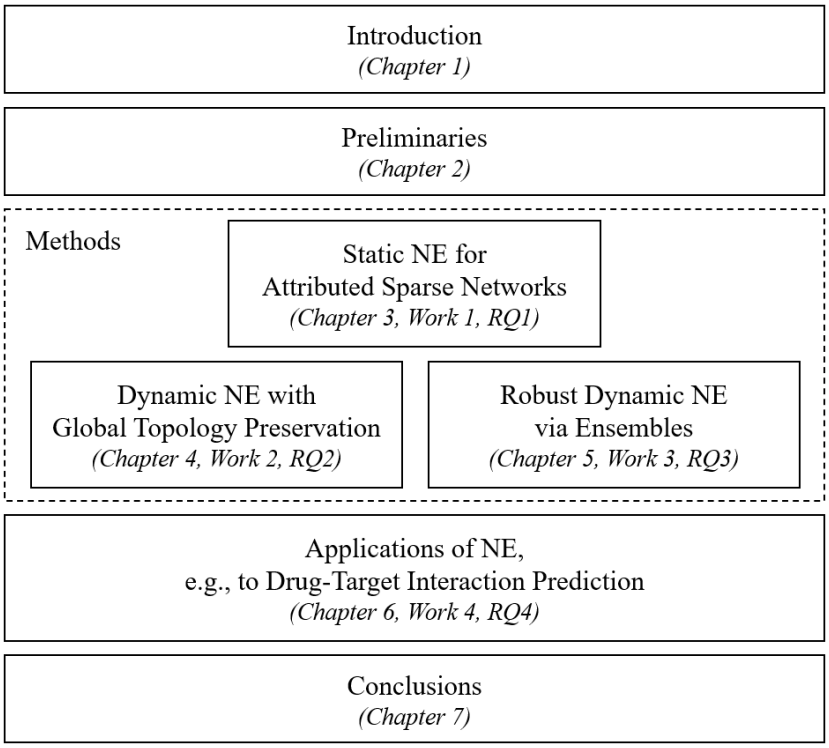


Figure 1.5: The outline of the thesis.

In Chapter 1, we have introduced the motivation of NE, the research questions and the contributions of this thesis.

For the remaining chapters[1], Chapter 2 first revisits the literature of NE, and then introduces some common concepts for later chapters. In order to answer RQ1, RQ2 and RQ3, Chapter 3, 4 and 5 present Work 1, 2 and 3 respectively in great details. To answer RQ4, Chapter 6 first discusses the generic applications of NE, and then presents Work 4 in great details. In Chapter 7, we summarize the thesis and discuss future directions.

[1]Work 5 and 6 are briefly discussed in Chapter 7. Work 7 is briefly discussed in Chapter 3.

Chapter 2

Preliminaries

Network Embedding (NE) is a representation learning paradigm for networks to extract topological features. It has recently become a popular topic in the artificial intelligence and its related areas where the data is in the form of networks or graphs. Where does NE come from? What are the representative approaches of NE? How is the recent situation of NE? To answer these questions, we review the literature of NE via three stages. After that, we discuss why and how to develop from static networks to dynamic networks which are not such obvious to think about but are the focus in Chapter 4 and 5. Finally, we declare the notations and concepts used throughout this thesis.

2.1 Literature Review of Network Embedding

In this section, we review the literature of NE via three stages. Specifically, stage 1 recaps the related topics which greatly affect or motivate the later development of NE. Stage 2 categorizes and discusses the representative approaches of NE during its rapid development. Stage 3 analyzes the recent situation of NE.

Note that, we focus on NE to *static plain networks* in this section due to the following reasons. First, the primary objective of NE is to exploit network topology or structure, though there could be other objectives while applying NE to other types of networks. Second,

various types of NE methods are usually developed from static plain NE methods. And third, the more specific literature will be given in Chapter 3-6 respectively.

2.1.1 Stage 1: an early stage

Before NE becoming popular, i.e., roughly before year 2014, there are four related topics which greatly affect or motivate the later development of NE.

The first related topic is *manifold learning*, which aims to project the data points in high dimensional vector space into a low dimensional vector space (or low dimensional manifold) from which these data points are sampled [9]. There are several popular methods such as LLE [8], Isomap [65], and Laplacian Eigenmap [66]. They first construct a weighted graph, e.g., by applying k-nearest-neighbors to data points, and then obtain low dimensional embeddings while preserving the structure of the constructed graph.

Although these methods from manifold learning are proposed to deal with data points in high dimensional vector space instead of a real graph, the approach of obtaining embeddings while preserving the structure of the constructed graph can be also applied to NE. To be more specific, how they formulate problem as an optimization problem after the constructed graph is ready, and how to solve the optimization problem using *matrix factorization*, both together motivate some later NE works such as [67] and [68].

The second related topic is the *word embedding* technique from Natural Language Processing (NLP). The most representative method is Word2Vec with the Skip-Gram model [69]. It first employs a fixed-size window to slide along a given text. Then, it builds word training pairs by the center word and its neighboring words within a window. And finally, it feeds the training pairs to the Skip-Gram model, which is derived from the neural probabilistic language model [70]. Intuitively, the more frequently two words co-occur, the closer their embeddings are. The assumption of this idea is that similar words have similar contexts. After Word2Vec, many other word embedding techniques are proposed such as [71] and [72]. Note that, word embedding is now an essential step in various NLP tasks.

The word embedding technique motivates some later NE works mainly because of the

following analogy. If one conducts random walks on a network, the resulting node sequences are very similar to the word sequences generated from a text. This analogy motivates a series of NE works such as [73] and [74].

The third related topic is the *autoencoder* coming from deep learning [10, 75]. A conventional autoencoder tries to minimize the reconstruction error between input layer and output layer (the two layers have exactly the same number of neurons). As a result, the middle layer(s) with less neurons can obtain a compressed representation of the input layer. It can be naturally seen as a dimensionality reduction technique. Also thanks to the powerful fitting ability guaranteed by the universal approximation theorem [76], the autoencoder are employed in several later NE works such as [77] and [78].

The fourth related topic is the *Convolutional Neural Networks* (CNN) also coming from deep learning [10, 75]. CNN is often used for image data. The basic idea is to extract the local stationarity property from a given 2D sliding window [79]. As a matter of fact, an image is a special network where most pixels connect to their eight adjacent pixels. Regarding a general network, one can define the window as k-hops neighbors of a node, which leads to *Graph Convolutional Network* and may be seen as a kind of generalization of CNN [11]. This affects a lot of later NE (or NE related) works such as [20] and [21].

2.1.2 Stage 2: a rapidly developing stage

The *static plain* NE (or simply called as NE in this section) is developing rapidly from year 2014 to 2018. During this stage, a huge number of NE methods are proposed, which has been reviewed in several surveys [48–53] around 2018. In general, they can be categorized into four representative approaches.

2.1.2.1 Skip-Gram based NE Approach

The first breakthrough NE method towards large-scale networks is DeepWalk[1] [73], which belongs to Skip-Gram based NE approach. There are three key steps. First, it conducts

[1]DeepWalk receives over 6000 Google Scholar citations in December 2021.

truncated random walks over each node in a network to generate node sequences. Second, a fixed-size window is applied to slide along each node sequence to build node training pairs. And third, it feeds node training pairs to train the Skip-Gram model.

The Skip-Gram model [69] aims to predict the contextual nodes (or words in NLP) given its central node (or word) of the sliding window. It is a simple neural network model where the input layer takes the one-hot representation of a node; the fully connected middle layer without activation function is called project layer which acts as the simple index of a row of the weight matrix (i.e., node embedding matrix) between input and middle layers; and the fully connected last layer (yielding contextual embedding matrix between middle and last layers) adopts softmax to output the probability of the co-occurrence of central and contextual nodes. If the co-occurrence is observed from the sliding window, its maximizes the probability for this co-occurrence. Overall and intuitively, the closer two nodes in a node sequence are, the more training pairs of the two nodes would generate, and accordingly the closer embedding vector of the two nodes would be after training.

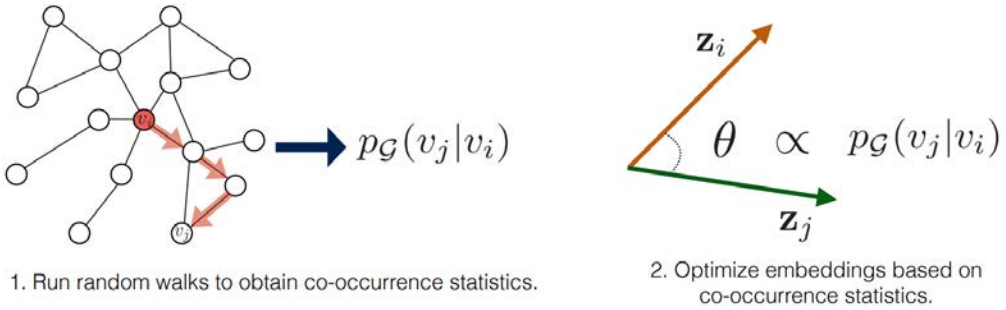


Figure 2.1: Skip-Gram based (or called random walk based) NE approach [48].

In this category, another representative work is Node2Vec [74]. It extends DeepWalk by modifying the first step with more flexible random walks so that one can balance between local topology and global topology. Instead of the above Skip-Gram model with softmax, it employs the Skip-Gram Negative Sampling model[2] to reduce computation costs due to softmax. Moreover, many other Skip-Gram based (or sometimes called random walk based) NE works can be found in surveys [48–53].

[2]The Skip-Gram Negative Sampling (SGNS) model treats the problem as a binary classification problem rather than a multiclass classification problem. The details of SGNS can be found in Chapter 4.

2.1.2.2 Matrix Factorization based NE Approach

The matrix factorization based NE approach stems from some manifold learning methods such as [8, 65, 66], which first construct a neighborhood graph, and then factorize the adjacency matrix of the graph to seek a lower rank approximation. As a graph is given in the NE problem, these manifold learning methods can be easily adapted to it. The general form of objective function for the matrix factorization based NE approach could be $\min ||\mathbf{S} - \mathbf{Z}^T\mathbf{Z}||_2^2$ where $\mathbf{S} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ is a matrix containing pairwise similarity measure between nodes among $|\mathcal{V}|$ nodes, and $\mathbf{Z} \in \mathbb{R}^{|\mathcal{V}|\times d}$ is the node embedding matrix [48]. In the context of manifold learning, once a neighborhood graph is constructed, it often sets $\mathbf{S}$ to the adjacency matrix $\mathbf{A}$ of the neighborhood graph.

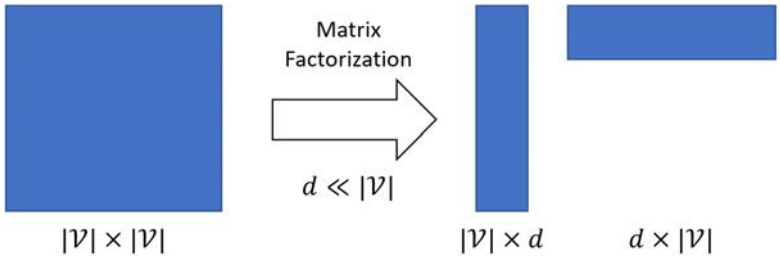


Figure 2.2: Matrix factorization based NE approach. $|\mathcal{V}|$ denotes the number of nodes. d denotes the dimension of embeddings.

Unlike directly using the adjacency matrix $\mathbf{A}$ that only reflects the first-order proximity or one-hop away neighbors, GraRep [67] and HOPE [68] also consider higher-order proximities between nodes. They first transform an adjacency matrix into its transition matrix $\mathbf{A}_{\text{transition}}$ (still one-hop relationship), and then obtain, e.g., k-hop relationship by matrix multiplication of the transition matrix $\mathbf{A}^k_{\text{transition}}$. Finally, they learn node embeddings with d dimensions by factorizing the new matrix, e.g., $\mathbf{S} = \mathbf{A}^k_{\text{transition}}$ with k-hop features, using the aforementioned objective function. Other methods in this route may encode more advanced topological information such as community patterns [80] and hierarchical patterns [64] into $\mathbf{S}$. For more matrix factorization based NE methods, the readers may refer to surveys [48–53].

2.1.2.3 Autoencoder based NE Approach

The autoencoder can be easily employed in NE problems, since it is originally designed to learn data representation or coding [81]. It minimizes the reconstruction error between input layer and output layer (the two layers have exactly the same number of neurons), and the middle layer(s) with less neurons can learn a compressed representation of the input layer. To be more specific, each row $\mathbf{S}_i \in \mathbb{R}^{|\mathcal{V}|}$ of a matrix $\mathbf{S} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ containing pairwise similarity measure between nodes is fed to train the autoencoder using the objective function, e.g., $\min ||\mathbf{S}_i - \hat{\mathbf{S}}_i||_2^2$ where $\mathbf{S}_i$ is the input of autoencoder and $\hat{\mathbf{S}}_i$ is the output of autoencoder. After training, we can feed $\mathbf{S}_i$ again to the trained autoencoder where the middle layer with d neurons produces the embeddings $\mathbf{Z}_i \in \mathbb{R}^d$ for node v_i.

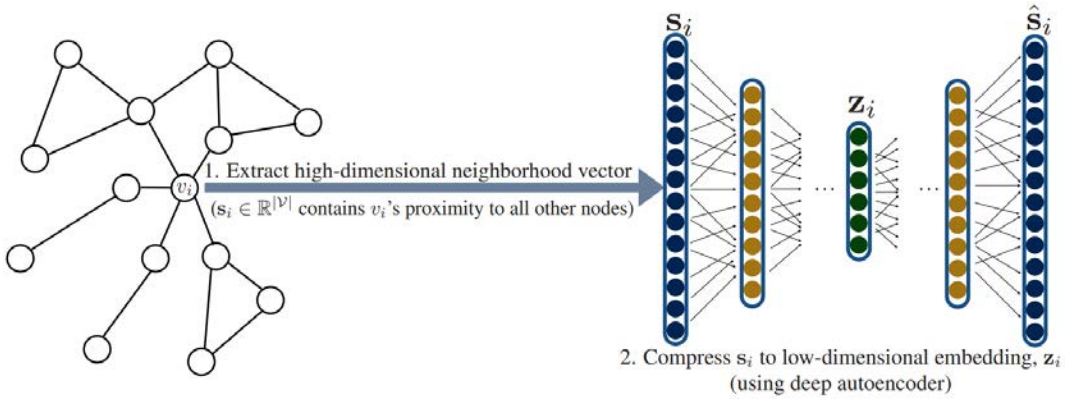


Figure 2.3: Autoencoder based NE approach [48].

Two representative works in this category are SDNE [77] and DNGR [78]. SDNE directly takes the adjacency matrix (set to be $\mathbf{S}$) of a network as the input of an autoencoder, so as to minimize the reconstruction error of each row of the adjacency matrix. Note that, each row of the adjacency matrix reflects the first-order proximity of each node to all nodes in the network. Unlike SDNE, DNGR takes the input of the pointwise mutual information matrix (set to be $\mathbf{S}$) DNGR [78], which offers more advanced network features after further transformation of the adjacency matrix. The readers may refer to surveys [48–53] for more autoencoder based NE methods.

2.1.2.4 Graph Convolution based NE Approach

The graph convolution based NE approach can be intuitively analogous to Convolutional Neural Networks (CNN), which has received a great success in computer vision. By treating an image as a special network formed by adjacent pixels, the analogy between CNN and graph convolution is conceptually illustrated in Figure 2.4.

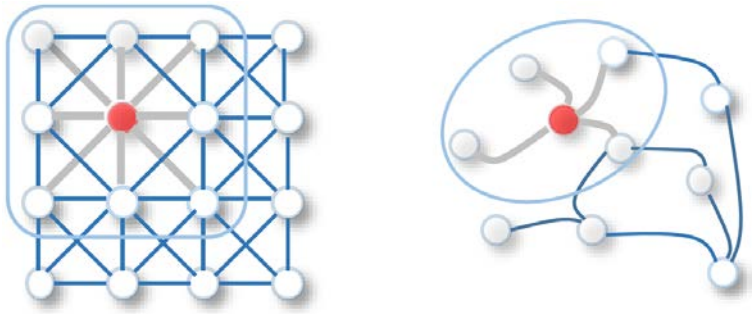


Figure 2.4: Graph convolution based NE approach [11]. The left is 2D convolution filter with size 3 by 3 over an image. The neighbors of a center node are in fixed size and ordered. The right is graph convolution filter with one-hop away neighbors over a network. The neighbors of a center node are in varying size and unordered.

One representative method is GraphSAGE [20]. It first defines neighbors of a node by one-hop away neighbors. Second, attributes from the node and its neighbors are aggregated, combined, and transformed as the new attributes (i.e., embeddings) for the node. Specifically, this step can be written as $\mathbf{h}_v^k = f_{\text{nonlinear}}^k(\mathbf{W}^k \cdot f_{\text{combine}}^k(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}_v}^k))$ and $\mathbf{h}_{\mathcal{N}_v}^k = f_{\text{aggregate}}^k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}_v\})$ where $\mathcal{N}_v$ defines neighbors of node v; $f_{\text{aggregate}}^k$ is an aggregation function (e.g., elementwise mean which is irrelevant to the order of neighbors); f_{combine}^k can be a simple concatenation operation between two vectors; $\mathbf{W}^k$ contains trainable parameters that are shared while training over all nodes; $f_{\text{nonlinear}}^k$ can be a Sigmoid function which finally yields new embeddings for node v at layer k; and $k \in \{1, ..., K\}$. For $\mathbf{h}_v^0$, one can assign node attributes to it for attributed networks or node degree to it for plain networks. Third, the parametric functions in the second step can be trained by a specific task, e.g., node classification. After training, we use the trained parametric functions to obtain the final embeddings from the last layer K. Note that, GraphSAGE belongs to the spatial based

approach as intuitively shown in Figure 2.4, while another representative method called GCN [21] belongs to the spectral based approach related to graph signal processing [82].

Moreover, the performance of graph convolution based NE methods depends on not only network topology but also node attributes. These methods are often employed to attributed networks, although they can also be employed to plain networks with artificial attributes [21]. Besides, graph convolution based NE approach is usually under a semi-supervised learning paradigm though we can also train it using an unsupervised learning paradigm [20], whereas previous three approaches are usually under an unsupervised learning paradigm. For more graph convolution based NE methods, the readers may refer to NE surveys [48–53] and the surveys specialized in graph neural networks [11, 83–85].

2.1.3 Stage 3: a well-established stage

After the rapid development from year 2014 to 2018, NE then becomes a well-established topic. From the academic perspective, several good surveys [48–53] have been published around 2018 to systematically review this topic. In stage 2 above, we have summarized four representative NE approaches, in which the most representative static plain NE methods of each approach are introduced. Since 2018, the research interest of NE has largely moved to more complicated networks such as attributed networks, dynamic networks, etc. The main reason might be that these networks, compared to the static plain network, are more realistic and remain more challenges while applying NE to them.

From the industrial perspective, world-leading companies, such as Facebook, Tencent, Google, Baidu, Amazon, and Alibaba, have paid considerable attention to the NE topic, because they own a huge number of network data such as social networks, knowledge graphs, and user-item networks. To better utilize these networks, they have done or support a surge of NE works such as the papers [86–91] and the related open-source softwares `https://github.com/facebookresearch/PyTorch-BigGraph`, `https://github.com/dmlc/dgl`, `https://github.com/PaddlePaddle/PGL`, `https://github.com/rusty1s/pytorch_geometric`, and `https://github.com/alibaba/graph-learn`. These evidences imply that NE would have

a big impact on our daily life, e.g., online social networks (Facebook and Tencent), web searching (Google and Baidu), and online shopping (Amazon and Alibaba).

In summary, starting from 2018, NE has been widely recognized as an important topic in both academia and industry.

- In academia, researchers start to propose NE methods for more complicated networks such as attributed networks and dynamic networks, which still remain many challenges.
- In industry, engineers and researchers start to integrate NE methods into industrial products and apply NE methods to specific real-world applications.
- It is also worth noticing, as a byproduct of NE, an emerging topic called Graph Neural Networks (GNN) start to attract much attention from both academia and industry. NE usually obeys an unsupervised paradigm to learn generic embeddings for various tasks, whereas GNN often follows a semi-supervised or supervised paradigm to learn task specific embeddings typically from attributed networks [88].

2.2 From Static Networks to Dynamic Networks

Most real-world networks are dynamic by nature, while embedding dynamic networks still remains many challenges. To tackle the challenges, this thesis proposes two NE methods for embedding dynamic networks. Unfortunately, dynamic networks might not be such obvious to think about compared to static networks. In this section, we thus develop the concept of dynamic networks, and clarify the type of dynamic networks used in the thesis.

Before talking about well prepossessed networks, let us have a look at the original dataset of networks. According to network data collections at `http://snap.stanford.edu/data`, `http://networkrepository.com/dynamic.php`, and `http://konect.cc`, the most widely used format is edge list, i.e., $\{(v_1, v_2), (...), (...), ...\}$ where (v_1, v_2) denotes an edge between node v_1 and v_2, and similarly each $(...)$ includes an edge between two nodes. There are two advantages of using edge list compared to other formats, e.g., adjacency matrix. First, it greatly reduces the storage space in computers for most real-world networks that are sparse

by nature. Second, we can easily append auxiliary information to an edge (v_1, v_2), e.g., append node attributes and edge attributes via $(v_1, v_2, v_{1\text{attr}}, v_{2\text{attr}}, edge_{\text{attr}})$.

If only time information is appended to edges, we obtain the edge list with timestamps or called edge streams, i.e., $\{(v_1, v_2, timestamp), (...), (...), ...\}$ where each timestamp records the occurrence time of each edge. In fact, it is the most widely used format to store the original dataset of dynamic networks according to the aforementioned network data collections. Note that, we do not consider node attributes and edge attributes for dynamic networks, as there still exist many challenges while embedding a simple dynamic network. We leave the more complicated dynamic networks to the future work.

Given the static network data, i.e., $\{(v_1, v_2), (...), (...)...\}$, we can simply connect all edges to establish a static network. It is easy to conduct graph related algorithms, e.g., random walks, over the static network, or directly feed its adjacency matrix to a computational model. However, given the dynamic network data, i.e., $\{(v_1, v_2, timestamp), (...), (...), ...\}$, it seems to be ambiguous and not that obvious to represent the dynamic network.

There are quite a few approaches to represent a dynamic network as discussed in [92, 93] in the literature of network science, while there are two approaches widely used in embedding dynamic networks according to the very recent surveys [94–98] appeared around 2021.

The first one is *continuous-time* approach. If we only consider the event of adding edges [94] as the most dynamic network datasets offered in the above network data collections, this approach can directly employ $\{(v_1, v_2, timestamp), (...), (...), ...\}$ to continuously record the timestamp of adding edges. This approach does not require much preprocessing effort, and preserves all available information provided by the original dynamic network dataset. Nevertheless, it might not good at reflecting the network topology at a certain timestamp, though the time resolution is perfectly preserved.

The second one is *discrete-time* approach. This approach is better at reflecting the network topology, as well as preserves partial time information. Concretely, all edges are first ordered by time, and are then divided into slices by, e.g., fixed time interval or fixed number of edges. By doing so, multiple timestamps would be merged into one timestep, which yields the larger time granularity or window to show the pattern of network topology. After

the discretization from *continuous timestamps* to *discrete timestep*, we have edge streams $\{(v_1, v_2, timestep), (...), (...), ...\}$ as the edge view shown in Figure 2.5-a) and as the network view shown in Figure 2.5-b).

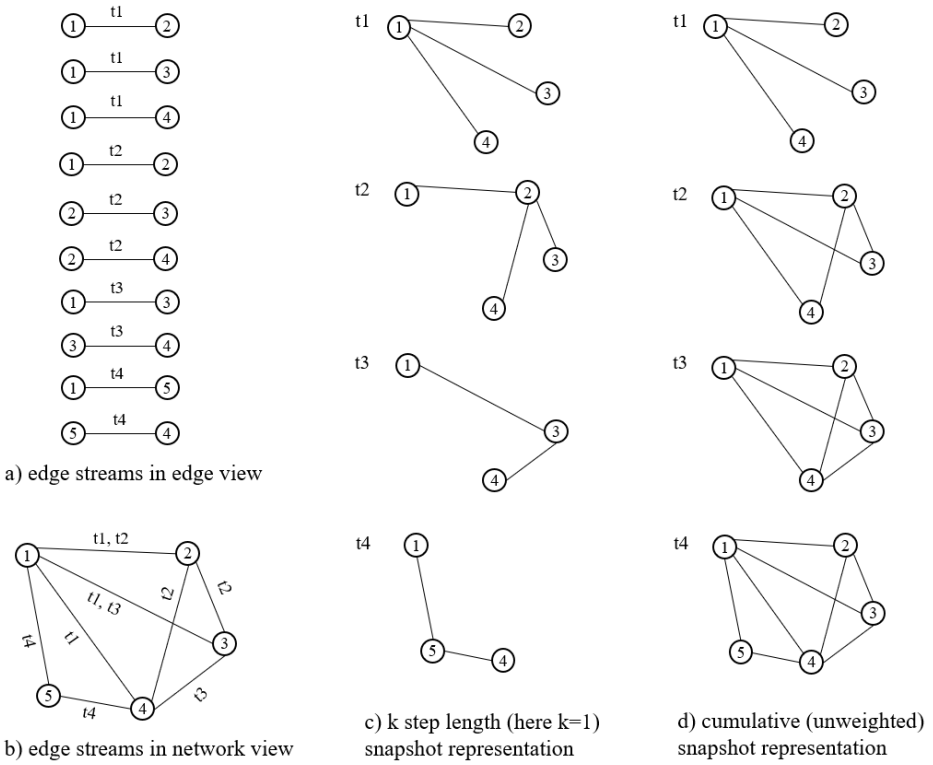


Figure 2.5: The discrete-time dynamic network. This thesis mainly considers d).

However, it is still unclear what the pattern of network topology should be fed to an embedding model at each timestep. To this end, we can expand the discretized edge streams over timesteps, which generates the snapshots taken at each timestep of a dynamic network. The snapshot could have two typical ways to capture edges as illustrated in Figure 2.5-c) and 2.5-d) respectively. Furthermore, the snapshot representation of Figure 2.5-d) is a special case of Figure 2.5-c), if we set a varying k to allow each snapshot to capture the edges from the very beginning to the current timestep.

It is noteworthy, in this thesis, we mainly consider the dynamic network generated in the same way used for generating Figure 2.5-d). The reasons are as follows. First, this type of dynamic network not only reserves partial time information but also better reflects the

evolution of network topology. Second, the evolution of network topology in Figure 2.5-d) is more stable and tractable than that in Figure 2.5-c).

In fact, finding a good way to generate a dynamic network as the input to a computational model is still an open problem. This problem is vital important especially in real-world applications, but it is out of the scope of this thesis. In general, we believe one key thing is the trade-off between *preserving time information* and *reflecting network topology* of the generated dynamic network, which would further affect the effectiveness and efficiency of its following computational model.

2.3 Notations and Concepts

Unless otherwise stated, this thesis generally employs the following notations.

- The uppercase letter, e.g., $\mathbf{A}$, denotes a matrix.
- The i-th row of matrix $\mathbf{A}$ is denoted as $\mathbf{A}_i$, which is a vector.
- The element in i-th row and j-th column of $\mathbf{A}$ is denoted as A_{ij}, which is a scalar.
- The lowercase letter, e.g. d, is a scalar or scalar variable.
- The number of elements in a set is given by $|\mathcal{V}|$.

Next, we clarify some important concepts used in this thesis.

- Networks and Graphs: They refer to the same thing, which at least consists of a set of nodes and a set of edges between the nodes.
- Topology, Structure, and Layout: The topology or topological information of a network only cares about the relationship (or connection) among nodes, while the structure or structural information of network additionally considers the strength of relationship (or weight of connection) among nodes. Networks with different layouts can have the same topology or the same structure, as both topology and structure do not need to concern about the absolute position of nodes.

- Proximity and Similarity: In the context of network embedding, the proximity is a widely used measurement of the similarity between nodes. The first order proximity reveals one-hop or immediate neighbors of a node, the second order proximity reveals two-hop or neighbors of neighbors of a node, etc [68]. For a pair of nodes in a network, they may have multiple proximities, but the smallest one (given by a shortest path algorithm) often dominates others. Apart from the similarity measured by the proximity, there exist other similarity measurements over a network.

- Network Embedding, Graph Embedding, Network Representation Learning, and Graph Representation Learning: In general, they all refer to the same thing. We use Network Embedding (NE) consistently throughout the thesis for better readability.

- Network Embedding (NE) and Graph Neural Networks (GNN): NE usually obeys an unsupervised paradigm to learn generic embeddings for various tasks, whereas GNN often follows a semi-supervised or supervised paradigm to learn task specific embeddings typically from attributed networks [88]. As both learn embeddings, GNN can be seen as a special NE approach, while NE can be an important step to GNN.

- Complex Systems, Datasets, Networks, Network Embedding, and Applications: 1) A complex system consists of many entities which involve interactions, relationships, or dependencies with each other. 2) A relational dataset is generated from the complex system. 3) We build a network based on the relational dataset, since we hope to utilize the relational information. 4) Network embedding can be adopted to extract features especially network topological features, and encode them into embeddings. 5) The embeddings can then facilitate some applications in the complex system.

Chapter 3

Static Network Embedding for Attributed Sparse Networks

As discussed in previous chapters, the early works of network embedding mainly focus on the static plain networks. Among these static plain network embedding works, the relatively earlier works aim to preserve microscopic network topologies, e.g., the proximity between nodes, while the relatively latter works try to additionally capture macroscopic network topologies. As one of the relatively latter works, we also proposed a static plain network embedding method called HSRL[1] to additionally preserve the *hierarchical* topology, a kind of macroscopic network topology, of a network. For the readers who are interested in HSRL, we refer them to our paper [64] for more details.

The aforementioned (static) plain networks only provide topological information. As a result, network embedding methods can only utilize the topological information. In fact, many real-world networks also offer node attributes, e.g., the profile of users in social networks. This type of networks is called (static) attributed networks under a static environment.

[1]Specifically, HSRL or Hierarchical Sampling Representation Learning recursively [64] compresses an input network into a series of smaller networks using a community-awareness compressing strategy. Then, any existing network embedding method can be used to learn node embeddings for each compressed network. Finally, the node embeddings of the input network are obtained by concatenating the node embeddings resulting from each compressed network.

Based on our work [58], this chapter presents a static network embedding method for the attributed network, especially for the topological sparse attributed network. This chapter is intended to answer the RQ1 of the thesis "*how can we embed an attributed sparse network effectively, efficiently, and robustly?*". The source code to reproduce this work is available at `https://github.com/houchengbin/OpenANE`

The organization of this chapter is as follows. Section 3.1 discusses the background and motivation of this work. Section 3.2 reviews the prior related works of learning node embeddings for attributed networks. Section 3.3 first introduces a new generic embedding framework that allows to integrate different sources of information to learn enhanced node embeddings. Based on the generic embedding framework, Section 3.3 then develops the proposed method and derives its time and space complexity. Extensive empirical studies of six downstream tasks on seven datasets are reported and discussed in Section 3.4. Finally, the chapter summary is presented in Section 3.5.

3.1 Background

Attributed networks are powerful data representation for many real-world complex systems (e.g., a social network with user profiles) in which entities (or users) can be represented as nodes; the interaction or relationship between entities can be represented as edges; and the auxiliary information of entities (or user profiles) can be represented as node attributes [48–50]. Comparing to a plain network, an attributed network has better capability to describe a complex system, since it not only reflects the topological information of the system but also records the attribute information of the entities in the system.

To facilitate various network inference tasks such as link prediction [17, 99, 100] and node classification [14, 20, 101], Network Embedding (NE) is proposed to learn low-dimensional node embeddings while persevering one or more network properties [48–50], so that the off-the-shelf distance metrics or machine learning algorithms can be effectively and efficiently applied. The early works [67, 68, 73, 74, 77, 102] only consider topological information while learning node embeddings, but cannot utilize attribute information which might also

be helpful in network inference tasks [5].

Attributed Network Embedding (ANE), which aims to learn enhanced node embeddings by considering both topological and attribute information, is attracting much attention [14–19]. The assumption behind ANE is that network topology and node attributes can provide complementary information [18] to jointly improve the performance of network inference tasks. This assumption can also be observed from human decision making process. For a PhD student who starts to seek a research topic, the common strategies of exploring related papers are to look at the references of some important papers (i.e., network topology), and to search for the papers with similar title, keywords, and etc. (i.e., node attributes). For a Facebook social network user, the user can decide to accept or reject the friend recommendation based on mutual friends (i.e., network topology) and user profiles such as attended university (i.e., node attributes). There are a lot of such examples. Therefore, it is reasonable to consider both topological and attribute information while learning node embeddings.

Nevertheless, the existing ANE methods still face some challenges. First, most existing ANE works have not carefully considered the issues caused by *sparse networks*, and accordingly proposed the special treatment to explicitly tackle the issues. As a result, they may obtain poor node embeddings on sparse or extremely sparse networks, especially for those methods [15, 17, 19–22] relying on edges to process node attributes, since the sparse network with less edges would restrict attributes processing. The sparse networks are important real-world scenarios [23] such as some incomplete networks, e.g., crawling a paper citation network in arXiv Computational Complexity subfield for recent five years, and the snapshots of some dynamic networks at early stages. The sparse network is a relative term and can be quantified by the ratio between the number of edges and the linear or the quadratic number of nodes [103]. The concrete examples of sparse networks, e.g., defined as $\frac{edges}{nodes} < 5$, can be found at Koblenz Network Collection [104], in which one can also find the extreme sparse networks, e.g., defined as $\frac{edges}{nodes} < 1$, with some isolated nodes via extracting the snapshots of some dynamic networks at early stages.

Second, the *scalability* of ANE problem becomes a more challenging problem compared with plain network embedding problem, since the methods to solve ANE problem require

additional computational resources to also utilize node attributes. A few very recent ANE works [105–108] that could also handle sparse networks, unfortunately require either $O(n^2)$ or $O(n^3)$ where n is the number of nodes, to utilize node attributes. Note that, it should be scalable in terms of not only the time complexity, but also the space complexity. For example, if the space complexity is $O(n^2)$, a network with 10^6 nodes at least requires $8 \cdot 10^{12}/1024^3 \approx$ 7500 Gigabytes memory if each floating-point number occupies 8 bytes.

Third, it is worth noticing the *robustness* of ANE methods to different networks or the same network with different sparsities regarding parameter tuning. Instead of grid search for the best hyper-parameter(s) [14, 16, 17, 20, 21], the robustness is desirable to design a generic ANE method. To be more specific, it should have relatively good and stable performance for few typical choices or by following explicit tuning advice [24], so that others can easily apply it in different real-world applications without much effort on hyper-parameter tuning, especially if it already has several other hyper-parameters to tune [25].

To tackle the above challenges, we first propose a generic embedding framework that allows to integrate different sources of information together to learn enhanced node embeddings. In other words, the information fusion process and the network embedding process are two cascaded steps in turn, which is different from most previous methods that they either adopt the reverse order of the two steps or mix the two steps together. The such design (i.e., information fusion first and then network embedding) aims to explicitly cope with *sparse networks* and increase the *robustness* of network embedding process. After that, several carefully selected techniques based on the criteria of *scalability* such as ball-tree k-nearest neighbors technique and random walks based Skip-Gram embedding technique, are employed to realize the proposed ANE method.

The main contributions of this work are as follows.

- We carefully discuss the issues caused by sparse networks for the different categories of network embedding methods.
- Based on a new generic embedding framework, a robust and scalable ANE method is realized to effectively embed attributed sparse (or extremely sparse) networks, and the

novel idea is to learn node embeddings upon the reconstructed denser network after information fusion via transition matrices.

- The theoretical complexity analysis shows the scalability of the proposed ANE method, and the extensive empirical studies (6 tasks, 7 datasets, 8 methods[2]) demonstrate the effectiveness, efficiency, and robustness of the proposed ANE method.

3.2 Prior Related Work

The first category of related works is Plain Network Embedding (PNE), which can only make use of network topology. DeepWalk [73] employs truncated random walks to obtain node sequences, which are then fed into Skip-Gram language model [69] to embed nodes closer if they co-occur more frequently. Node2Vec [74] extends DeepWalk by using more flexible truncated walks to capture network topology. To learn node embeddings, LINE [102] and SDNE [77] preserve both the first and second order proximities, while HOPE [68] preserves higher proximities. N-NMF [109] and ComVAE [110] consider not only microscopic structure (e.g., proximities between nodes) but also macroscopic structure (e.g., communities).

Although using the high order proximites [68, 77, 102] may relieve the sparsity issue, it cannot solve this issue caused by missing edges, since an edge indicating the first order proximity between two nodes is often assumed to have a stronger relationship than the higher order proximity. Furthermore, the high order proximity is built on the existence of the first order proximity. Therefore, it becomes harder to build high order proximities if the input network becomes sparser, and it is impossible to build any order proximities to a node isolated from all other nodes. Unlike these PNE methods, we directly establish the first order proximity between two nodes based on the similarity of their node attributes.

The second category of related works is Attributed Network Embedding (ANE), which utilizes both network topology and node attributes to learn enhanced node embeddings. The works in this category might be divided into four sub-categories. 1) *Matrix Factorization*

[2]The unified framework of the source code including over ten embedding methods is open sourced at `https://github.com/houchengbin/OpenANE` for benefiting future research in network embedding area.

based approach: TADW [14] and AANE [16] encode topological and attribute information into two matrices, and then formulate the ANE problem as a bi-convex optimization problem (the problem is convex for one variable if another variable is fixed) to jointly learn node embeddings. ANEM [111] encodes microscopic proximity structure, macroscopic community structure, and node attributes into three matrices, and then formulate the ANE problem as a multi-convex optimization problem (the problem is convex for one variable if remaining variables are fixed) to jointly learn node embeddings. One main challenge is the scalability, as they finally need to perform matrix factorization [15]. 2) *Graph Convolution based approach*: GCN [21] and graphSAGE [20] first define node neighbors based on network topology, and then aggregate attributes of neighboring nodes for further computing. These methods heavily rely on edges to aggregate node attributes, and hence may obtain less accurate node embeddings for sparse networks. More recently, GRCN [105] first combines the attribute pairwise similarity matrix with the adjacency matrix, and then applies either GCN or graphSAGE based on the combined matrix. It can handle the network sparsity issue, however, explicitly obtaining the attribute pairwise similarity matrix requires $O(n^3)$. Besides, there are a large number of other methods in this direction as discussed in a recent survey [11]. 3) *Deep Neural Networks based approach*: ASNE [17] encodes node ID and attributes together as the input, and node co-occurrence probabilities as the output, of a deep neural networks model. Recently, there are several other methods [18, 106, 107, 112–114] in this direction. They all employ auto-encoders to preserve some network proximities and/or integrate different sources of information. These methods might be good at mining nonlinearity, but the computational complexity is relatively high or hard to estimate, which is related to the number of nodes or edges, the architecture of neural networks, and the way to generate training samples. It is worth noticing that [107] and [106] can handle network sparsity issue, since they first combine the (attribute or polarity) pairwise similarity matrix with the adjacency matrix, and then adopts auto-encoders for further processing. However, explicitly obtaining the attribute or polarity pairwise similarity matrix at least requires $O(n^3)$ or $O(n^2)$ respectively. 4) *Random Walks based approach*: TriDNR [15] (ignore label part) and SANE [19] jointly maximize the likelihood of preserving network topology and node attributes. They

adopt random walks based PNE method for the part of preserving network topology, but another part of using node attributes still rely on the node pairs sampled by the random walks. The sparse network may not well utilize node attributes, since random walks in a sparse network lead to less combination of node pairs. Besides, [22] and [115] aim to embed the attributed network with binary and/or sparse attributes. They would encounter the similar problem due to the limitation of random walks on a (topological) sparse network.

The proposed ANE method belongs to random walks based ANE approach, so that it inherits the scalability [15, 19]. Unlike previous works in this direction, the information fusion and network embedding are two cascaded steps in turn, so as to cope with possible sparse networks by reconstructing an enriched denser network before network embedding step. Besides, unlike [105–107], our method does not need to explicitly obtain the attribute pairwise similarity matrix, which is computational expensive. Finally, this work focuses on unsupervised ANE methods, since the aim is to learn generic node embeddings for various downstream tasks which may or may not have labels.

3.3 Method

We first introduce a generic embedding framework, based on which the proposed method is developed. After that, we present the formal problem definition, the details of the proposed method, and its implementation with complexity analysis.

3.3.1 A Generic Embedding Framework

In this section, we introduce a new generic embedding framework that allows to integrate different sources of information to learn enhanced embeddings. Note that, this framework does not require the original data to be networked, though the proposed network embedding method is developed based on this framework.

Considering a generic embedding problem which can be treated as a generalized problem of network embedding, the real-world entities can be seen as data points where each data

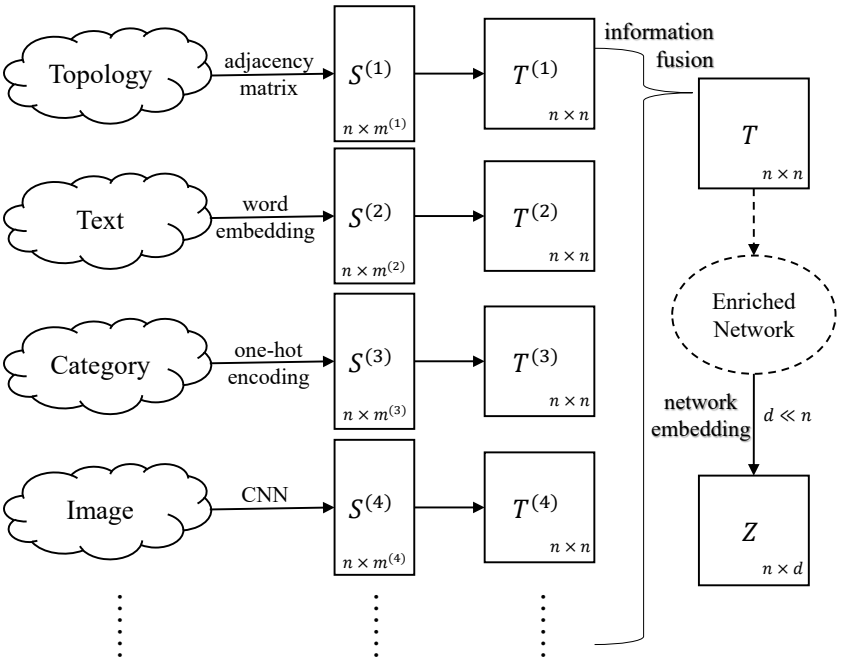


Figure 3.1: A new generic embedding framework.

point i has p sources of information to describe it, i.e., $\mathcal{S}_i = \{\mathbf{S}_i^{(1)}, \ldots, \mathbf{S}_i^{(p)}\}$. The p different sources of information of each data point may come from its network topology, textual data, categorical data, image data, and so on. As shown in Figure 3.1, there are different ways to convert different sources into matrices where each row vector $\mathbf{S}_i^{(q)} \in \mathbb{R}^{m^{(q)}}$ denotes the vector representation of data point $i \in \{1, ..., n\}$ from $q \in \{1, ..., p\}$ source. For example, one may use Convolutional Neural Networks (CNN) to extract the features from the image source of data point i, and Figure 3.1 presents one possible technique for prepossessing each source information. After the raw data prepossessing, we can obtain matrix $\mathbf{S}^{(q)} \in \mathbb{R}^{n \times m^{(q)}}$ for each source of information.

To integrate different sources of information and then smoothly employ a network embedding method to learn data embeddings, one possible solution is to use the *transition matrix*, i.e., a square matrix with non-negative real numbers and each row summing to one. As shown in Figure 3.1, it consists of two main cascaded steps.

For the former step, i.e., information fusion, it first transforms each matrix $\mathbf{S}^{(q)}$ into its transition matrix $\mathbf{T}^{(q)} \in \mathbb{R}^{n \times n}$ where each entry (i, j) reflects the pairwise similarity of data

points. Intuitively, if (i, j) are more similar than (i, k) at q source, the transition probability of (i, j) should be larger at the q source. After the transformation, the information fusion is achieved via a linear combination among all sources of information, i.e.,

$$\mathbf{T} = \sum_{q=1}^{p} \lambda^{(q)} \mathbf{T}^{(q)} \quad \text{s.t.} \quad \sum_{q=1}^{p} \lambda^{(q)} = 1 \tag{3.1}$$

where $q \in \{1, ..., p\}$ denotes different sources of information, and the factor $\lambda^{(q)} \in [0, 1]$ is a non-negative real number. There are two main reasons of using such linear combination. First, $\mathbf{T}$ is still a transition matrix given each $\mathbf{T}^{(q)}$ is a transition matrix. The proof is in Appendix A.1. Second, it is easy to interpret and/or adjust the importance of each source contributing to the final result. One can even inject expert knowledge to $\lambda^{(q)}$ if needed.

It might be interesting to notice that, the transition matrix $\mathbf{T}$ (a.k.a. stochastic matrix or Markov matrix) in Eq. (3.1) can be seen as a mixture of Markov chains. Intuitively, we have totally p sources of information and $\lambda^{(q)}$ gives the probability to choose the q-th source from p sources. We then go to the Markov chain based on $\mathbf{T}^{(q)}$.

For the latter step, i.e., network embedding, it either explicitly reconstructs the enriched network based on transition matrix $\mathbf{T}$, or implicitly treat $\mathbf{T}$ as a network. And then, any existing PNE methods [67, 68, 73, 74, 77, 102] can be applied to this plain network $\mathbf{T} \in \mathbb{R}^{n \times n}$ to learn node embeddings. Each row vector of embedding matrix $\mathbf{Z} \in \mathbb{R}^{n \times d}$ where $d \ll n$ is the node embedding vector corresponding to the original data point or real-world entity.

3.3.2 Problem Definition

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{W}, \mathbf{X})$ be a given attributed network where $\mathcal{V} = \{v_1, \ldots, v_n\}$ denotes a set of $|\mathcal{V}|$ or n nodes; $\mathcal{E} = \{e_{ij}\}$ denotes a set $|\mathcal{E}|$ edges; the weight attached to each edge is a scalar W_{ij}; the attributes associated to each node are in a row vector $\mathbf{X}_i$; and $i, j \in \{1, \ldots, n\}$ are subscripts. Note that, the proposed ANE method can accept either directed or undirected and either weighted or unweighted attributed networks.

Definition 1: Topological Information Matrix $\mathbf{W} \in \mathbb{R}^{n \times n}$. The topological information refers to network linkage information, which is encoded in matrix $\mathbf{W}$. There are several popular

choices to encode topological information [68] such as the first-order proximity that gives the information of one-hop neighbors, the second-order proximity that gives the information of two-hop neighbors, etc. In this work, the first-order proximity is used to define $\mathbf{W}$, i.e., the adjacency matrix, since it can be directly obtained without further computation.

Definition 2: Attribute Information Matrix $\mathbf{X} \in \mathbb{R}^{n \times m}$. The attribute information refers to network auxiliary information attached to each node, which is encoded in matrix $\mathbf{X}$ where each row $\mathbf{X}_i \in \mathbb{R}^m$ corresponds to the node attributes for node v_i. To obtain the vector presentation $\mathbf{X}_i$, one may employ word embedding if it is textual auxiliary information [15], one-hot encoding if it is categorical auxiliary information [17], and so on.

Definition 3: Attributed Network Embedding (ANE). It aims to find a mapping function f such that $\mathbf{Z} = f(\mathbf{W}, \mathbf{X})$ where $\mathbf{Z} \in \mathbb{R}^{n \times d}$, $d \ll n$, and the row vector $\mathbf{Z}_i \in \mathbb{R}^d$ is the node embedding for node v_i. The pairwise similarity between node embeddings should reflect the pairwise similarity between nodes in the original input attributed network considering both network topology and node attributes.

3.3.3 Method Description

Based on the above generic embedding framework in Section 3.3.1 and the problem definition in 3.3.2, we now present the proposed ANE method namely Robust and Scalable Attributed Network Embedding or RoSANE. Sections 3.3.3.1, 3.3.3.2, and 3.3.3.3 address how to obtain the transition matrices of network topology and node attributes in terms of scalability, how to achieve information fusion using Eq. (3.1), and how to employ PNE method to efficiently learn node embeddings, respectively. Finally, Section 3.3.4 summarizes the algorithm implementation of RoSANE, and derives its time and space complexity.

3.3.3.1 Preprocessing Transition Matrix

As discussed in Section 3.3.1, we need to obtain the transition matrices for the two sources of information of an attributed network, i.e., network topology and node attributes. Concretely, it requires to transform topological information matrix $\mathbf{W} \in \mathbb{R}^{n \times n}$ and attribute information

matrix $\mathbf{X} \in \mathbb{R}^{n \times m}$ into topological transition matrix $\mathbf{T}^{(W)} \in \mathbb{R}^{n \times n}$ and attribute transition matrix $\mathbf{T}^{(X)} \in \mathbb{R}^{n \times n}$ respectively. Note that, such transformation should be scalable in terms of both time and space complexity. And each entry in the transformed transition matrices should reflect the pairwise similarity of the corresponding pair of nodes.

For the topological transition matrix $\mathbf{T}^{(W)}$, each entry in the topological information matrix $\mathbf{W}$, i.e., the weight of edge naturally reflects the pairwise similarity of the corresponding pair of nodes. In order to inherit such desirable meaning from $\mathbf{W}$, the topological transition matrix $\mathbf{T}^{(W)}$ can be obtained by

$$\mathbf{T}_i^{(W)} = f_{\text{norm}}(\mathbf{W}_i) = \frac{\mathbf{W}_i}{\sum_{j \in n} W_{ij}} \tag{3.2}$$

where f_{norm} is a function operating on row vector $\mathbf{W}_i$ such that each row of $\mathbf{W}$ sums to one, i.e., a discrete probability distribution. But $\mathbf{T}^{(W)}$ might not be a strict transition matrix, since the isolated node leads to all-zero row. One may assign an uniform distribution to each all-zero row as what Google Matrix does [116], however, we retain them to avoid meaningless edges. Note that, all-zero rows are fixed in Eq. (3.4) using attribute information.

For the attribute transition matrix $\mathbf{T}^{(X)}$, one naive solution is to calculate the pairwise similarity for all possible pairs of rows of $\mathbf{X}$, which gives an n-by-n pairwise similarity matrix, and then analogously apply Eq. (3.2) to obtain $\mathbf{T}^{(X)}$. Unfortunately, the resulting n-by-n pairwise similarity matrix (often with few zero entries) needs to store almost (n^2) nonzero entries, which is not scalable in terms of memory usage. Besides, the resulting $\mathbf{T}^{(X)}$ is also an extreme dense transition matrix, which leads to an almost fully connected network where two very dissimilar nodes would also have an edge. In fact, it is not necessary to explicitly calculate the pairwise similarity of all possible pairs, since the aim is to *enrich the topology of the input (sparse) network using the top-k most attribute similar nodes* so that the attribute dissimilar nodes are omitted. As a result, there are only (kn) nonzero entries in $\mathbf{T}^{(X)}$, which make it scalable in terms of memory usage by storing it in a sparse matrix format.

Finding the top-k most attribute similar nodes now becomes a K-Nearest Neighbors (KNN) search problem. Due to the large number of nodes and the high dimensional node attributes in many real-world scenarios, ball-tree KNN technique is employed [117]. The

basic steps are to first partition data points into a series of nesting hyper-spheres which construct a binary tree data structure, and then query k nearest neighbors of a given data point based on the binary tree [118]. It is worth noting that the ball-tree KNN technique is often based on a valid distance metric, e.g., Euclidean distance, whereas the generic embedding framework requires a similarity measure, e.g., Cosine similarity. Therefore, after ball-tree KNN, we need the following transformation

$$\cos(\mathbf{X}_i, \mathbf{X}_j) = 1 - \frac{||\mathbf{X}_i - \mathbf{X}_j||^2}{2} \quad \text{s.t.} \quad ||\mathbf{X}_i|| = 1 \;\; \forall i \in \{1, ..., n\} \tag{3.3}$$

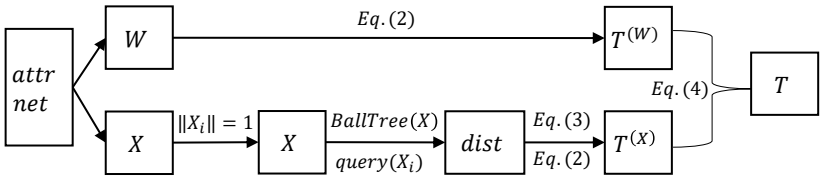


Figure 3.2: The block diagram of the general steps before network embedding.

3.3.3.2 Information Fusion

The Eq. (3.1) can be used to enrich the topology of input (sparse) network using top-k most attribute similar nodes, i.e., the information fusion process. However, as discussed in Section 3.3.3.1, the topological transition matrix $\mathbf{T}^{(W)}$ might not be a strict transition matrix due to the possible isolated nodes, and thus the (attributed) biased transition matrix $\mathbf{T}$ might not be a strict transition matrix. Instead of directly using Eq. (3.1), a trick is employed to guarantee the resulting biased transition matrix $\mathbf{T}$ being a strict transition matrix, i.e.,

$$\mathbf{T}_i = \begin{cases} \mathbf{T}_i^{(X)} & \text{if } \mathbf{T}_i^{(W)} \text{ is all zeros} \\ \lambda \mathbf{T}_i^{(W)} + (1-\lambda)\mathbf{T}_i^{(X)} & \text{otherwise} \end{cases} \tag{3.4}$$

where λ is the balancing factor of the importance between network topology and node attributes; the subscript i indicate the corresponding row of each transition matrix; and each row of $\mathbf{T}$ gives discrete probability distribution $\mathbf{T}_i \in \mathbb{R}^n$ to indicate the transition probability from node i to any nodes in the network. Furthermore, Eq. (3.4) can enrich a (sparse) network from two perspectives. Intuitively, for two nodes without an edge before, there will be a new edge (thus a denser network) if their node attributes are top-k similar. For two nodes already with an edge, the weight of this edge will be augmented (thus more discriminative) if their node attributes are top-k similar.

Based on the biased transition matrix $\mathbf{T}$, an enriched denser network can be reconstructed, which comes with several properties: 1) it reflects both topological and attribute information; 2) it is a weighted and directed network, which is more informative than an unweighted and undirected network; and 3) it does not contain isolated nodes, and each node in the reconstructed network gains enriched connectivities.

3.3.3.3 Learning Node Embeddings

The problem now becomes learning node embeddings on the reconstructed network (weighted, directed, and not attributed). Regarding scalability, a random walks based PNE method, i.e., a modified version of DeepWalk [73] is employed to learn node embeddings. Instead of unweighted random walks and softmax strategy used in DeepWalk, we adopt weighted random walks and negative sampling strategy [74].

Specifically, the weighted random walks with length l starting from each node in the network for r times, generate a set of $r|\mathcal{V}|$ node sequences. For each node sequence, a fixed-size window $w+1+w$ is used to slide along it. For each window, several training pairs (v_c, v_i) are generated such that v_c is the center node and v_i is one of other nodes inside the window. By doing so, we obtain a list of training pairs $(v_c, v_i) \in D$ for all sequences, which is then fed into Skip-Gram Negative Sampling (SGNS) model [69] for training node embeddings. For each pair (v_c, v_i), we maximize the following formula

$$\max \ \log \sigma(\mathbf{Z}_c \cdot \mathbf{Z'}_i) + \sum_{i'=1}^{q} \mathbb{E}_{v_{i'} \sim P_D}[\log \sigma(-\mathbf{Z}_c \cdot \mathbf{Z'}_{i'})] \tag{3.5}$$

where σ is the Sigmoid function; the operator $\cdot$ denotes a dot product between vectors; $\mathbf{Z}_c \in \mathbb{R}^d$ is the embedding vector from row c of the trainable embedding matrix $\mathbf{Z}$ (between input and middle layers a.k.a. a project layer), while $\mathbf{Z}_i \in \mathbb{R}^d$ and $\mathbf{Z'}_{i'} \in \mathbb{R}^d$ are the embedding vector from column i or i' of another trainable matrix $\mathbf{Z'}$ (between middle and output layer) [120]; $v_{i'}$ is the negative sample from the unigram distribution P_D [121]; and q is the number of negative samples [122]. The aim of maximizing formula (3.5) is to make embedding vectors *similar* if they co-occur, and *dissimilar* if they are negative samples. The overall objective is to sum over all $(v_c, v_i) \in D$, i.e.,

$$\max_{\mathbf{Z},\mathbf{Z'}} \sum_{(v_c,v_i)\in D} \{\log \sigma(\mathbf{Z}_c \cdot \mathbf{Z'}_i) + \sum_{i'=1}^{q} \mathbb{E}_{v_{i'}\sim P_D}[\log \sigma(-\mathbf{Z}_c \cdot \mathbf{Z'}_{i'})]\} \tag{3.6}$$

It can be optimized by stochastic gradient ascent over D to learn these trainable embeddings. Intuitively, the more frequently a pair of nodes $(v_c, v_i) \in D$ co-occurs[3], the more similar or closer their embeddings would be.

3.3.4 Algorithm and Complexity

The proposed ANE method consists of two cascaded steps: information fusion and network embedding. Their implementation details are summarized in Algorithm 1 and 2 respectively.

In Algorithm 1, first, lines 1-7 are all based on row vector operation, and hence each line of them is easily parallelized or vectorized. Second, line 2 aims to project $\mathbf{X}_i$ into a unit hyper-sphere, so that it enables line 5 employ Eq. (3.3) for transforming Euclidean distance into Cosine similarity. Third, the three transition matrices, with many zero entries, are stored and operated in Compressed Sparse Row (CSR) format to reduce space complexity.

In Algorithm 2, first, the two loops in lines 2 and 3 can be easily parallelized, as random walks are independent with each other. Second, instead of explicitly reconstructing a network for running random walks, the Alias sampling method is adopted to simulate random walks

[3]In the experiment, we found that the quality of node embeddings not only depends on the co-occurrence statistics, but also the order of training pairs. It suggests that a random order obtains better node embeddings comparing with a fixed order, since the random order of training pairs may help the optimizer jump out some bad local optima.

Algorithm 1 Information Fusion Step of RoSANE

Input: topological information matrix $\mathbf{W}$, attribute information matrix $\mathbf{X}$, balancing factor λ, top-k value k

Output: attributed biased transition matrix $\mathbf{T} \in \mathbb{R}^{n \times n}$

obtain $\mathbf{T}^{(W)}$ by Eq. (3.2)
normalize $\mathbf{X}$ by $||\mathbf{X}_i|| = 1 \; \forall \; i$
construct BallTree($\mathbf{X}$)
query k nearest neighbors of $\mathbf{X}_i \; \forall \; i \in \{1, ..., n\}$, and return *dist* with *index*
transform Euclidean *dist* to Cosine *sim* by Eq. (3.3), and return unnormalized $\mathbf{T}^{(X)}$
obtain normalized $\mathbf{T}^{(X)}$ analogously by Eq. (3.2)
obtain $\mathbf{T}$ by Eq. (3.4)
return attributed biased transition matrix $\mathbf{T}$

Algorithm 2 Network Embedding Step of RoSANE

Input: attributed biased transition matrix $\mathbf{T}$, walks per node r, walk length l, window size w, negative samples q, embedding dimensionality d

Output: node embedding matrix $\mathbf{Z} \in \mathbb{R}^{n \times d}$

initialize a list $walks = [\,]$
for $iter \in r$ **do** ▷ parallel
    for $v_i \in \mathcal{V}$ **do** ▷ parallel
        initialize a list $walk = [v_i]$
        for $walk\_iter \in l$ **do**
            $curr\_node = walk[-1]$
            $prob\_dist = \mathbf{T}_{curr\_node}$
            $next\_node$ = AliasSampling($prob\_dist$)
            append $next\_node$ to $walk$
        append $walk$ to $walks$
obtain $\mathbf{Z} \in \mathbb{R}^{n \times d}$ by training word2vec($walks, w, q, d$)
return node embedding matrix $\mathbf{Z}$

based on the discrete probability distribution given by $\mathbf{T}_i$, which can greatly reduce the time complexity [74]. Third, the widely-used Word2Vec [122] approach with Eq. (3.5), as described in Section 3.3.3.3, is employed to train embeddings by the analogy of the nodes in a walk and the words in a sentence [73].

Recall that $|\mathcal{V}| = n$ is the number of nodes, $|\mathcal{E}| = \theta n$ is the number of edges where θ is the average degree of a network; m is the number of attributes of each node; and r, l, w, k, q, and d are user-defined constants. Next, we will investigate the scalability of RoSANE in terms of time and space complexity.

In Algorithm 1, the most time-consuming operations are in lines 3 and 4, whose complexities are $O(n(\log(n))^2)$ and $O(n\log(n))$ respectively [123]. Besides, because all transition matrices are stored and operated in CSR format, the complexities of lines 1-2 and 5-7 are $O(\theta n)$, $O(mn)$, $O(kn)$, $O(kn)$, and $O(\theta n + kn)$ respectively. In Algorithm 2, Alias sampling method in line 8 needs $O(rln)$ for repeating rl times on n nodes [74], and the time complexity of whole network embedding process (as discussed in Section 3.3.3.3) requires $O(rlwdn)$ [73, 74]. In summary, the overall time complexity is $O(n(\log(n))^2 + n\log(n) + \theta n + mn + kn + rlwdn)$. Since r, l, w, d and k are the user-defined constants, plus θ and m are the constants of a given attributed network, it can be rewritten as $O(n(\log(n))^2 + n\log(n) + n)$. Note that, even if there are $n = 10^{18}$ nodes, we have $\log(n) \approx 60$ and $(\log(n))^2 \approx 60^2 \ll n = 10^{18}$, and hence RoSANE is scalable in terms of time complexity for very large-scale networks.

For space complexity, several space-consuming variables needed to store in memory are $\mathbf{W}$, $\mathbf{X}$, $\mathbf{T}^{(W)}$, BallTree($\mathbf{X}$), $\mathbf{T}^{(X)}$, $\mathbf{T}$, look-up table of Alias sampling, training pairs D, weights of SGNS model $\mathbf{Z}'$ and $\mathbf{Z}$. The overall space complexity of them is $O(\theta n + mn + \theta n + mn + kn + (\theta + k)n + 3n + rlwn + dn + dn)$. Note $\mathbf{W}$, $\mathbf{T}^{(W)}$, $\mathbf{T}^{(X)}$, and $\mathbf{T}$ are in CSR format. After eliminating the constants, it can be rewritten as $O(n)$, which demonstrates the scalability of RoSANE in terms of space complexity.

3.4 Experiments

3.4.1 Experimental Settings

3.4.1.1 Datasets

Six *real-world* attributed networks and a series of *synthetic* attributed networks are employed to study the effectiveness, efficiency, and robustness of the proposed method RoSANE. To be specific, MIT and UIllinois are two Facebook friendship social networks for each university, and there are seven properties attached to each Facebook user: status flag, gender, major, second major, dorm, year, and high school [124]. The 'year' is taken out as the label, and the remaining six properties are converted into attributes via one-hot encoding. Moreover, Cora, Citeseer, Pubmed, and DBLP are four paper citation networks. The attributes of Cora and Citeseer preprocessed by [125] are in binary vector form, whereas the attributes of Pubmed and DBLP prepossessed by [125] and [15] respectively are in continuous vector form. The well preprocessed attributed networks are provided at `https://github.com/houchengbin/NetEmb-Datasets`, and their statistics are summarized in Table 3.1.

Table 3.1: The statistics of six real-world attributed networks.

Datasets	Nodes	Edges	Attributes	Labels
Cora	2708	5429	1433	7
Citeseer	3327	4732	3703	6
MIT	5298	217118	2804	32
Pubmed	19717	44338	500	3
UIllinois	26803	1073235	2921	34
DBLP	60744	52890	400	4

3.4.1.2 Compared Methods

All the methods compared in the experiments are in *unsupervised* fashion, i.e., no label is used during embedding. Except DeepWalk and attrSVD, all other methods utilize both topological and attribute information.

- DeepWalk [73]: It is a random walks based PNE method that only utilizes topological information, i.e., $\mathbf{W} \in \mathbb{R}^{n \times n} \mapsto \mathbf{Z} \in \mathbb{R}^{n \times d}$.
- attrSVD: It only utilizes attribute information by applying SVD for dimensionality reduction [126] on the attribute information matrix, i.e., $\mathbf{X} \in \mathbb{R}^{n \times m} \mapsto \mathbf{Z} \in \mathbb{R}^{n \times d}$.
- TADW [14]: It jointly models attribute and topological information as a bi-convex optimization problem under the framework of matrix factorization.
- AANE [16]: It is similar to TADW, but the problem is solved in a distributed manner.
- sageMean [20]: It aggregates attributes from neighboring nodes, and then takes the element-wise mean over them. For fair comparison, we adopt its unsupervised version.
- sageGCN: GCN first proposed by [21] is designed for semi-supervised learning. For fair comparison, we adopt the unsupervised generalized version of GCN by [20].
- ASNE [17]: It encodes node ID and attributes together as the input, and node co-occurrence probabilities as the output, of a deep neural networks model. The objective is to maximize the co-occurrence probability of the co-occurred node pairs generated by one-hop node neighbors.

3.4.1.3 Evaluation

Based on the original implementation, all compared methods are unified in one framework namely OpenANE, which enable us to use the same I/O and evaluate performance in a fair environment. Regarding hyper-parameters, we mostly follow the suggestions by the original papers. To be more specific, node embedding dimensionality $d = 128$ for all methods. For DeepWalk and RoSANE, walks per node $r = 10$, walk length $l = 80$, window size $w = 10$, negative samples $q = 5$, and top-$k = 30$ when needed. For sageGCN, sageMean, and ASNE, learning rate, dropout rate, batch size, and epochs are set to {search: 1e-2, 1e-3, 1e-4}, 0.5, 128, and 100 respectively. For TADW, AANE, ASNE, and RoSANE, the balancing factors are set to 0.2, 0.05, 1.0, and 0.2 respectively. For other hyper-parameters, please refer to OpenANE at `https://github.com/houchengbin/OpenANE`

Table 3.2: Datasets used in different experiments.

Datasets	LP	NC	Vis.	PR	Sen.	Time
Cora	✓	✓	✓		✓	✓
Citeseer	✓	✓			✓	✓
MIT	✓	✓	✓		✓	✓
Pubmed	✓	✓			✓	✓
UIllinois	✓	✓			✓	✓
DBLP	✓	✓		✓	✓	✓
Synthetic						✓

We conduct six types of experiments. The datasets used in different experiments are summarized in Table 3.2. Specifically, the sections 3.4.2.1-3.4.2.6 report the results of Link Prediction (LP) task, Node Classification (NC) task, Visualization (Vis.) task, Paper Recommendation (PR) case study, Sensitivity Analysis (Sen.), and Wall-Clock Time (Time) respectively. Except Vis. and PR, all other experiments are repeated 12 times and their arithmetic averages are reported.

3.4.2 Results and Discussions

3.4.2.1 Link Prediction

Link Prediction (LP) task is intended to predict missing links or potential links. We randomly remove 20% edges as the positive samples, and randomly generate the equal number of non-existing edges as the negative samples, together serving as the *ground truth* of testing set to evaluate LP tasks [17, 74]. Specifically, the LP task asks whether node pairs in testing set should have edges. The decision is made based on the cosine similarity between the corresponding pairs of node embeddings, and the final result is given by AUC score [17]. Furthermore, to study the effectiveness and robustness of network embedding methods for sparse networks, we further randomly remove different percentages of edges. The percentages of the removed edges for different sparsities are shown in the header of Table 3.3. Note that, only the remaining edges would be used during the upstream embedding phase.

Table 3.3: Left: AUC score of link prediction. Right: Micro-F1 score of node classification.

removed	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
					Cora-LP								**Cora-NC**			
attrSVD	79.52	79.71	79.64	79.96	79.89	79.89	79.80	79.74	72.24	72.55	72.13	72.96	72.47	72.34	72.69	72.60
DeepWalk	80.89	77.36	72.96	67.41	62.09	57.79	54.54	52.64	78.37	75.95	72.62	66.11	60.14	51.32	42.25	34.53
TADW	85.46	83.97	82.81	79.32	75.39	71.38	65.85	59.46	82.48	81.01	79.79	77.82	75.68	73.33	70.25	66.81
AANE	81.20	81.06	81.21	81.01	80.31	80.74	80.05	79.64	70.99	69.81	69.85	70.39	71.13	70.80	69.91	70.11
sageGCN	85.21	83.60	80.35	76.71	73.54	68.99	64.48	61.22	71.50	69.99	65.95	62.23	56.93	51.21	46.87	40.56
sageMean	76.46	73.05	70.56	68.63	64.63	61.71	57.82	53.34	62.46	58.12	54.46	49.96	46.32	41.38	37.25	34.79
ASNE	69.74	70.17	69.68	69.39	69.42	69.06	69.30	69.21	47.62	46.25	45.43	45.69	45.19	44.21	44.68	44.42
RoSANE	**92.99**	**92.38**	**91.91**	**90.28**	**88.70**	**87.68**	**85.95**	**83.59**	**84.32**	**83.28**	**82.17**	**80.53**	**79.00**	**77.94**	**74.98**	**72.86**
					Citeseer-LP								**Citeseer-NC**			
attrSVD	87.88	88.09	88.14	87.81	88.02	87.64	87.77	87.98	68.28	68.16	68.42	68.19	68.52	68.77	68.00	68.07
DeepWalk	69.97	68.13	65.45	63.00	61.90	59.17	57.67	54.51	53.43	49.60	46.56	42.67	39.64	33.51	29.22	25.21
TADW	86.88	84.78	83.12	80.08	76.91	71.92	67.23	62.22	71.47	70.58	69.96	69.25	67.91	66.72	65.38	63.49
AANE	88.63	88.29	87.89	88.00	88.20	88.08	87.65	87.43	69.25	69.44	69.81	69.52	69.00	68.96	69.25	**69.63**
sageGCN	87.20	85.63	84.30	81.81	78.46	76.10	71.67	67.27	54.70	52.07	50.77	47.62	44.47	41.74	37.63	34.22
sageMean	84.32	82.56	80.63	77.39	74.62	69.72	64.74	58.93	50.98	48.81	44.74	42.51	38.40	35.48	31.68	29.11
ASNE	77.09	76.60	75.46	75.37	75.26	73.54	74.46	74.16	43.39	44.02	42.00	42.33	40.34	41.21	41.90	41.69
RoSANE	**94.90**	**95.11**	**94.35**	**94.30**	**93.45**	**92.93**	**92.03**	**91.01**	**71.95**	**71.52**	**71.07**	**70.29**	**70.02**	**69.74**	**69.80**	69.22
					MIT-LP								**MIT-NC**			
attrSVD	64.66	64.55	64.51	64.66	64.62	64.69	64.63	64.80	35.02	35.18	35.26	34.99	34.76	34.79	34.77	35.28
DeepWalk	**91.80**	**91.32**	**90.78**	**89.88**	**88.65**	**86.75**	**83.48**	**76.40**	**79.53**	**78.70**	**77.35**	**76.34**	74.52	71.26	66.63	55.79
TADW	87.05	86.89	86.84	86.46	86.02	84.96	83.18	69.31	66.99	65.30	65.82	63.72	61.88	60.07	56.66	50.05
AANE	65.55	65.29	64.76	64.47	64.32	64.01	63.71	62.41	36.41	36.03	36.50	36.31	35.84	36.03	35.59	35.42
sageGCN	80.99	80.95	80.69	80.25	80.24	79.55	77.64	70.81	60.78	59.63	59.25	57.44	57.16	55.46	52.45	44.15
sageMean	73.76	74.02	73.44	73.75	73.35	72.70	65.02	58.07	53.78	53.16	51.09	51.65	49.89	48.20	41.33	32.23
ASNE	73.85	72.98	72.22	71.46	70.19	68.88	67.27	65.10	53.85	51.54	49.11	45.89	41.41	37.23	33.50	31.18
RoSANE	90.26	89.78	89.07	88.14	86.81	84.73	81.42	74.71	79.07	78.27	77.19	76.09	**74.60**	**71.71**	**67.34**	**58.67**
					Pubmed-LP								**Pubmed-NC**			
attrSVD	90.34	90.25	90.24	90.21	90.20	90.33	90.20	**90.16**	83.44	83.40	83.42	83.64	83.42	83.35	83.37	83.47
DeepWalk	78.42	77.18	75.95	74.26	72.18	69.76	66.17	60.93	76.13	73.50	69.97	66.14	62.05	56.88	50.82	43.52
TADW	90.17	89.53	88.55	86.87	84.82	81.53	76.00	67.60	**86.39**	**86.35**	**86.03**	**85.69**	**85.22**	**85.14**	**84.49**	83.70
AANE	84.06	84.11	84.26	84.30	84.35	84.69	85.10	86.18	85.00	85.07	84.90	84.93	85.00	84.87	84.85	**84.91**
sageGCN	92.17	92.08	91.21	90.14	89.82	87.24	84.97	80.83	81.10	80.19	79.50	78.05	78.38	75.79	74.40	72.28
sageMean	88.41	87.72	87.78	86.83	84.28	81.60	76.12	63.43	79.55	78.05	77.81	76.72	73.93	73.03	69.36	64.79
ASNE	79.51	78.54	76.46	74.18	72.22	69.47	65.22	61.45	78.81	77.82	77.46	77.20	76.72	76.02	74.24	71.03
RoSANE	**95.20**	**94.86**	**94.36**	**93.82**	**93.08**	**92.14**	**90.83**	88.71	83.59	83.04	82.48	81.88	81.13	80.24	79.83	78.75
					UIllinois-LP								**UIllinois-NC**			
attrSVD	61.30	61.36	61.34	61.31	61.29	61.32	61.27	61.36	45.94	45.92	45.91	45.90	45.76	46.03	45.82	45.84
DeepWalk	**94.62**	**94.30**	**93.95**	**93.50**	**92.75**	**91.43**	88.82	81.79	**80.27**	79.80	79.24	78.41	76.77	74.73	71.05	62.06
TADW	86.93	86.85	84.26	84.86	78.04	74.67	76.06	79.71	70.13	69.78	69.06	68.50	67.41	66.06	63.58	59.84
AANE	61.09	61.55	61.62	61.38	61.06	60.42	60.65	60.53	47.33	46.85	46.16	46.02	45.91	46.26	45.92	45.48
sageGCN	86.19	86.05	86.09	85.71	85.42	85.02	84.21	80.98	64.55	64.52	64.64	63.42	62.07	61.22	59.32	52.66
sageMean	85.43	84.85	84.76	84.89	84.27	83.56	82.77	78.76	63.81	63.23	62.35	62.65	60.67	58.70	56.47	47.21
ASNE	80.81	79.78	78.50	76.99	75.75	75.18	74.64	72.74	55.26	54.68	53.50	52.05	50.00	48.30	46.64	44.94
RoSANE	94.44	94.14	93.81	93.30	92.58	91.30	**88.98**	**83.30**	80.18	**79.87**	**79.30**	**78.61**	**77.25**	**75.59**	**73.03**	**65.74**
					DBLP-LP								**DBLP-NC**			
attrSVD	72.64	72.57	72.66	72.49	72.48	72.53	72.74	72.59	69.52	69.65	69.55	69.52	69.67	69.61	69.46	**69.54**
DeepWalk	93.57	92.47	91.48	89.68	87.29	84.03	78.80	69.10	52.33	51.78	51.03	50.19	49.04	47.52	45.63	42.89
TADW	69.43	68.48	67.64	65.53	63.41	61.27	58.58	56.00	72.71	72.28	72.01	71.57	71.09	70.35	69.35	68.32
AANE	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
sageGCN	90.45	89.50	88.32	86.77	84.28	81.65	76.48	69.18	71.72	71.22	70.67	70.29	69.75	68.85	68.27	66.96
sageMean	84.54	83.67	81.66	79.38	75.78	69.57	62.36	54.33	69.98	69.33	68.79	68.46	67.69	66.31	65.56	64.15
ASNE	67.83	65.62	65.17	64.07	62.74	61.46	62.10	64.70	65.64	65.68	65.49	65.61	65.31	65.67	65.70	65.73
RoSANE	**94.69**	**93.88**	**92.92**	**91.54**	**89.60**	**86.75**	**82.21**	**74.87**	**72.82**	**72.65**	**72.25**	**71.95**	**71.51**	**70.86**	**70.24**	69.13

The values (in decimal) in the first line show different percentages of edges removed. These edges will not be used during embedding. All the results (given by arithmetic average over 12 runs) are reported in %, and the best ones are marked in bold.

The left-hand side of Table 3.3 shows the results of 6 datasets under the different percentages of removed edges in LP tasks. First, the overall observation is that RoSANE obtain either the best results or the second best results for all networks with all different sparsities, which verifies its *effectiveness* in LP tasks. Second, comparing among the ANE methods (ignore attrSVD and DeepWalk) to see how well they can utilize two sources of information, RoSANE always achieves the best results for all cases. Third, as the same network becomes sparser (from the left to the right), the gain of RoSANE w.r.t. the second best ANE method often increases, e.g., 7.70% in Citeseer-LP-0.2, 12.49% in Citeseer-LP-0.5, and 23.74% in Citeseer-LP-0.9, w.r.t. sageGCN. Therefore, these observations demonstrate the *robustness* of RoSANE to different networks or the same network with different sparsities. The reasons might be two-folds: 1) RoSANE adopts two cascaded steps so that information fusion step gives an enriched denser network, which makes the network embedding step more robust; 2) The enriched denser network contains two sources of information, which enables the network embedding step to capture the high-order proximity (beyond pairwise relationship) of not only topological information but also attribute information.

Besides, one interesting observation is that DeepWalk (only uses topological information) and attrSVD (only uses attribute information) achieve the best results in some cases. In particular, DeepWalk achieves the best results in the most cases of social networks, i.e., MIT and UIllinois, due to the very rich topological information (see Table 3.1 for edges-nodes ratio). In these cases, the majority of useful attribute information (indicated by attrSVD) might be contained in topological information (indicated by DeepWalk), and parts of two sources of information might be conflicted with each other. As a result, ANE methods may receive worse results than DeepWalk or attrSVD. Nevertheless, RoSANE always outperforms other ANE methods for the reasons as discussed above.

3.4.2.2 Node Classification

Node Classification (NC) task aims to assign known label(s) to nodes without label(s). We randomly pick 50% nodes with labels for training a one-vs-rest logistic regression classifier. The remaining nodes with labels serve as the *ground truth* of testing set, and the final result

is given by Micro-F1 score [20] based on the prediction of the trained classifier. Similarly as LP tasks, different percentages of edges are removed to study the effectiveness and robustness of RoSANE in NC tasks.

The right-hand side of Table 3.3 shows the results of 6 datasets under the different percentages of removed edges in NC tasks. The overall observations in NC tasks are similar as that in LP tasks as discussed above. There are two main differences. First, for the social networks, i.e., MIT and UIllinois, RoSANE in general receives better results in NC tasks, whereas DeepWalk receives better results in LP tasks. Second, on Pubmed dataset, TADW obtains the best results in NC tasks, whereas RoSANE obtains the best results in LP tasks. In fact, due to the complicated real-world scenarios, different methods, tasks, and sources of information, all have different bias, e.g., topological information (indicated by DeepWalk) might be more important than attribute information (indicated by attrSVD) in LP tasks, and the opposite situation can be found in NC tasks. From an industrial perspective, RoSANE is worth trying, since RoSANE (with the fixed recommended hyper-parameters) in general obtains the most promising results than other embedding methods considering different datasets, sparsities, and tasks.

3.4.2.3 2D Visualization

Visualization task further reduces the dimensionality of node embeddings to 2D by Principal Component Analysis technique. We then assign different colors to nodes based on their labels. This task paves a way to intuitively understand why node embeddings can benefit downstream tasks like LP and NC tasks. The four most competitive methods, i.e., RoSANE, DeepWalk, sageGCN, and TADW (according to Table 3.3) are selected for visualizing Cora and MIT datasets. The upstream embedding settings are identical to Cora-LP/NC-0.2 and MIT-LP/NC-0.2 (LP and NC have the same settings under same dataset and sparsity).

As shown in Figure 3.3-a), RoSANE presents more distinguishable boundaries between different labels on Cora dataset comparing to other methods, which is consistent with the quantitative results in LP and NC tasks. Similarly, for Figure 3.3-b), RoSANE and DeepWalk both obtain superior visualization results than TADW and sageGCN on MIT dataset.

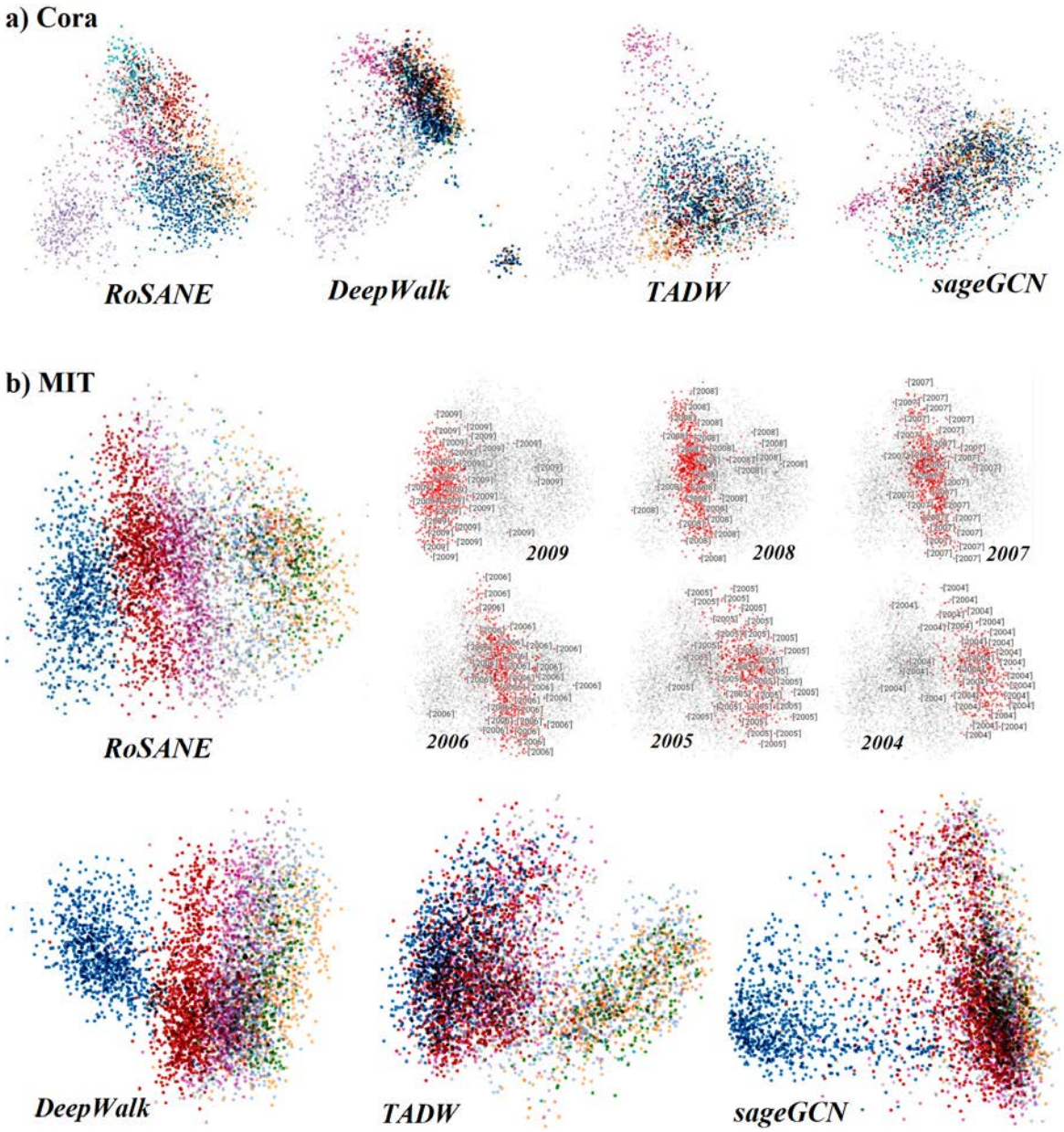


Figure 3.3: 2D visualization of node embeddings for a) Cora and b) MIT. Different colors indicate different labels. In particular, b) also highlights the result given by RoSANE of six corresponding years, which shows the temporal trend.

Furthermore, according to the six highlighted sub-figures of Figure 3.3-b), we observe an interesting phenomenon by RoSANE called *temporal trend*. It is in accordance with human common sense that students in a university have very close relationships if they graduate (or enter) in the same year. And more interestingly, students also have relatively closer relationships if they graduate in nearby years (usually within 4 years). However if the year window is more than 4 years, the students graduated in 2009 have a big gap to those students graduated in 2005 and 2004.

One may argue that DeepWalk is competitive with RoSANE on MIT dataset. In fact, if we carefully examine their visualization result, RoSANE might be better at distinguishing the nodes with less links (non-blue and non-red colors corresponding to the earlier years of MIT Facebook users) compared with DeepWalk. For the visualization result of DeepWalk on Cora, it is obvious that there are some nodes from different classes squeezed together. Besides, for the quantitative results in Table 3.3, we observe that RoSANE dramatically outperforms DeepWalk on the sparser networks, e.g., Cora, Citeseer, and Pubmed, although DeepWalk is comparable with RoSANE on the relatively denser networks, e.g., MIT and UIllinois. Moreover, the gain of RoSANE to DeepWalk continuously increases, as the same network becomes sparser. These observations indicate that RoSANE can effectively utilize nodes attributes to tackle the sparse networks or the sparse parts of a network.

3.4.2.4 Paper Recommendation: A Case Study

Thanks to the availability of raw paper titles of the DBLP citation network, we conduct a case study of paper recommendation, i.e., recommend similar papers for a given paper based on the resulting node/paper embeddings. This task is used to study some characteristics of RoSANE in recommendation related tasks. In this task, the upstream embedding settings are identical to DBLP-LP/NC-0.2. The downstream recommendation task is based on the cosine pairwise similarity between the embeddings learned by RoSANE.

Given an interested paper, e.g., "Dimensionality Reduction for Similarity Searching in Dynamic Databases" to retrieve (denoted as R paper), the returned top-10 papers are shown in Table 3.4 based on the top-10 highest scores. For R paper, there are three key terms:

Table 3.4: Paper recommendation results by RoSANE

Top	Paper
R	*Dimensionality Reduction for Similarity Searching in Dynamic Databases*
1	A Simple Dimensionality Reduction Technique for Fast Similarity Search in Large Time Series Databases
2	A non-linear dimensionality-reduction technique for fast similarity search in large databases
3	Fast Time-Series Searching with Scaling and Shifting
4	On the Effects of Dimensionality Reduction on High Dimensional Similarity Search
5	Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases
6	Dimensionality Reduction and Similarity Computation by Inner Product Approximations
7	A Comparison of DFT and DWT based Similarity Search in Time-Series Databases
8	Interval-Focused Similarity Search in Time Series Databases
9	Fast Time Sequence Indexing for Arbitrary Lp Norms
10	VQ-index: an index structure for similarity searching in multimedia databases

dimensionality reduction, similarity search, and dynamic database. For the recommended papers, the top-(1∼6, 10) are directly cited by R paper, whilst the top-(7∼9) are not directly cited by R paper. Among those directly cited papers, the top-1 has three matched terms. The top-(2,4,5,6) have two matched terms. The top-(3,10) have one matched term, and the top-3 cites three papers that are also cited by R paper, i.e., the second-order proximity, whereas the top-10 has zero such paper. Among those not directly cited papers, they all cite several papers that are also cited by R paper. The top-(7,8) have two matched terms, whereas the top-9 has one term 'Lp-norm' related to 'similarly search'.

It should be noted that, a paper recommendation system might not be a good system, if it only returns the papers cited by the retrieved paper. Because other papers that have many second-order (and higher order) proximities or have a very similar title are also worth recommending. This case study illustrates that RoSANE effectively utilizes both network topology and node attributes for paper recommendation.

3.4.2.5 Parameter Sensitivity

This section investigates the hyper-parameter sensitivity of RoSANE. Concretely, there are two special hyper-parameters λ for balancing the two sources of information and k for returning the top-k most attribute similar nodes. We conduct both LP and NC tasks on all six real-world datasets. The experimental settings are identical to dataset-LP/NC-0.2 as used in Table 3.3, except that Figure 3.4-a) varies λ and Figure 3.4-b) varies k.

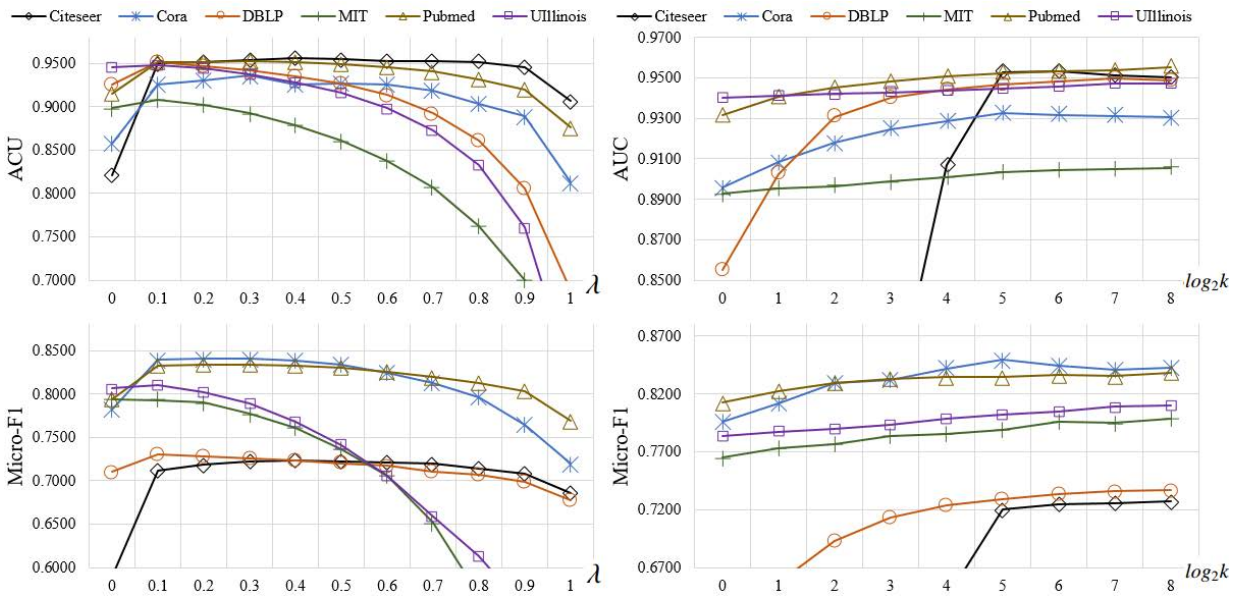


Figure 3.4: Parameter sensitivity: a) the left two figures for $k = 30$ and varying λ; b) the right two figures for $\lambda = 0.2$ and varying top-k (x-axis in $\log_2 k$ scale)

For Figure 3.4-a), $k = 30$ is fixed, but λ is varied from 0.0 to 1.0 with step 0.1. The results of six different real-world datasets all show that RoSANE can obtain satisfactory performance for λ with different values around 0.2, which indicates that RoSANE does not require much effort on tuning λ. In fact, all the above experiments fix $\lambda = 0.2$ and RoSANE already obtains the most promising results, although it is still possible to further improve the performance if one has priori knowledge. For example, MIT and UIllinois have very rich topological information and hence set $\lambda = 0.1$ to obtain better performance (by doing so, RoSANE may outperform DeepWalk as shown in Table 3.3). Second, for the two extreme cases $\lambda = 0.0$ (only uses topology) and $\lambda = 1.0$ (only uses attributes), RoSANE often

obtains relatively worse results, which implies the usefulness of employing the two sources of information to learn embeddings.

For Figure 3.4-b), $\lambda = 0.2$ is fixed, but k is varied from 2^0 to 2^8 with exponential step 1. It shows that the performance of RoSANE increases as k increases when k is relatively small, and then is saturated when k becomes larger. A smaller k indicates that the less attribute similar nodes are used to enrich the later reconstructed network, but it requires less time to retrieve similar nodes. As a trade-off, $k = 30$ might be a good typical choice[4] and is used in all above experiments. In summary, this section conveys that RoSANE can *robustly* obtain satisfactory performance with the *fixed* hyper-parameters $\lambda = 0.2$ and $k = 30$ (and hence requires less effort on hyper-parameter turning) at least for the six tested networks.

3.4.2.6 Wall-Clock Time

The wall-clock time, excluding I/O and downstream tasks, is extracted from the experiments of the removed 20% edges (in both LP and NC tasks for simulating different sparsities) as shown in Table 3.3. The experiments are conducted in a computer cluster with the same fixed budget for all methods: 36 cores of Intel Xeon E7-8880v3 CPU, 256G memory, Nvidia Kepler K80 with 12G GPU memory (sageGCN, sageMean, and ASNE use GPU for acceleration), CentOS-6.5 64-bit. The wall-clock time results of ANE methods on the six real-world datasets are shown in Table 3.5 for comparing their time efficiency[5]. First, RoSANE consumes either the shortest time or acceptable time, which confirms the time efficiency of RoSANE. Second, the wall-clock time of sageGCN, sageMean, and ASNE significantly increases as the number of edges increases, since they rely on edges to process node attribute as discussed before. Finally, the wall-clock time of TADW, AANE, and RoSANE is highly related to the number of nodes, however, TADW and AANE are less efficient than RoSANE due to the explicit matrix factorization operation [15].

[4]In addition to fixing $\lambda = 0.2$ and varying k, we have also tested several different λs. The general tendency remains the same as that of $\lambda = 0.2$.

[5]The results of wall-clock time of ANE methods are based on their original implementations with different levels of code optimization that will also affect the results to some extent. Besides, DeepWalk and attrSVD are not included in Table 3.5, as they belong to a different class of methods than ANE methods.

Table 3.5: The wall-clock time of ANE methods (in seconds) on the six real-world attributed networks for comparing their time efficiency.

nodes	edges		TADW	AANE	sageGCN	sageMean	ASNE	RoSANE
2708	5429	Cora	102	42	96	90	40	**33**
3327	4732	Citeseer	155	56	213	187	**43**	48
5298	217118	MIT	197	116	5963	5256	1735	**113**
19717	44338	Pubmed	540	1570	317	299	361	**265**
26803	1073235	UIllinois	2212	1883	30993	26468	11080	**801**
60744	52890	DBLP	7280	n/a	**370**	358	643	1144

To empirically study the scalability of RoSANE regarding the increasing number of nodes, edges, or attributes, we generate a series of synthetic attributed networks. The node attributes part is based on the random binary vector with dimensionality m, and the network topology part is based on the Erdos-Renyi graph [127, 128] in which any possible edge has a fixed probability p_e to present. Generating an Erdos-Renyi graph requires p_e and n (the number of nodes), but we are more interested in $|\mathcal{E}|$ (the number of edges) and n. For an undirected Erdos-Renyi graph, $p_e \approx \frac{2\theta n}{n(n-1)}$ where θ is the average degree, $|\mathcal{E}| = \theta n$, and $n(n-1)/2$ is the number of all possible edges of an undirected graph. As a result, it enables us to generate an Erdos-Renyi graph using θ (or $|\mathcal{E}|$) and n.

Table 3.6: The wall-clock time of RoSANE (in seconds) on synthetic attributed networks to empirically study its scalability w.r.t. the increasing number of nodes, edges, or attributes.

$SET1$	$\theta = 2^4$	$m = 2^7$								
n	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}
$T(n)$	7	12	21	38	74	149	346	886	2342	6940
$SET2$	$n = 2^{12}$	$m = 2^7$								
θ	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
$\lvert\mathcal{E}\rvert$	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}
$T(\theta)$	37	39	42	47	57	79	123	204	378	731
$SET3$	$n = 2^{12}$	$\theta = 2^4$								
m	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}
$T(m)$	39	39	40	39	39	40	44	56	77	115

Table 3.6 shows the wall-clock time of RoSANE on the synthetic attributed networks. All the experiments in Table 3.6 are conducted in the aforementioned hardware expect the memory is further limited to 16GB. There are three sets of experiments. For each set, we fix two variables and investigate the third one which increases in $\times 2$ rate. For SET2 and SET3, the wall-clock time $T(\cdot)$ increases in less than $\times 2$ rate within the tested range. The range is chosen based on the real-world scenarios: for SET2, the average degree θ is often no greater than 1000 (see real-world networks collection at Koblenz Network Collection [104]); for SET3, the dimensionality m is specified by users while encoding node attributes, which should not be too large in order to avoid the curse of the dimensionality. Moreover, regarding SET1, $T(n)$ increases in a rate no greater than $\times 3$ within the tested range. The choice of largest n is due to the given limited memory at 16GB (so that many computers can meet it). It is possible that the number of nodes n might be larger. However, the tendency indicates that the rate itself increases very slowly due to the log term in time complexity as discussed in Section 3.3.4. In summary, RoSANE is scalable to the number of nodes n, the average degree θ or number of edges $|\mathcal{E}|$, and the dimensionality of attributes m. The results in Table 3.6, to some extent, also empirically verify the complexity analysis in Section 3.3.4.

3.5 Chapter Summary

In this chapter, to answer the RQ1 of the thesis, we presented our work [58], which proposed a novel ANE method to effectively embed attributed sparse or extremely sparse networks.

Specifically, we first introduced a new generic embedding framework, which was intended to enrich a network from multiple sources of information. Based on the generic embedding framework, a robust and scalable attributed network embedding method called RoSANE was realized. RoSANE first integrated topological and attribute information via transition matrices so as to reconstruct an enriched denser network. It then learned node embeddings upon the enriched denser network. In above two steps, the techniques such as Ball-tree KNN and random walks based Skip-Gram model were adopted to guarantee the scalability, which was demonstrated via complexity analysis. The extensive empirical studies of various

downstream tasks on six real-world attributed networks with different sparsities as well as a series of synthetic attributed networks confirmed the effectiveness, efficiency, and robustness of the proposed method.

Our experiments, however, also showed that the two baseline methods DeepWalk (using network topology) and attrSVD (using node attributes) can slightly outperform all ANE methods (using both information) in some cases. This might be due to the less complementary information or the conflict between two sources of information. Therefore, one promising future direction is to investigate the interplay between network topology and node attributes before applying any ANE method. Another future work is to extend RoSANE to cope with the sparsity issue at the early stages of a dynamic network in the context of the dynamic network embedding problem [27, 129].

Chapter 4

Dynamic Network Embedding with Global Topology Preservation

Chapter 3 discussed the static network embedding method. In fact, most existing network embedding methods are designed for static networks. These methods try to deal with different challenges caused by different types of static networks (e.g., attributed networks) or different properties to preserve (e.g., hierarchical topology).

However, many real-world networks are dynamic by nature, e.g., new friendships would establish between existing users and/or new users in a social network, and devices would regularly connect to or accidentally disconnect from routers in a wireless sensor network. Although most challenges in embedding static networks have been addressed, there are still many challenges in embedding dynamic networks.

Based on our work [59], this chapter presents a dynamic network embedding method particularly for better global topology preservation, so as to answer the RQ2 of the thesis "*how can we embed a dynamic network with global topology preservation?*". Note that, we do not consider node attributes in this work. The source code to reproduce this work is available at `https://github.com/houchengbin/GloDyNE`

This chapter is organized as follows. Section 4.1 discusses the background of this work, especially why dynamic network embedding needs global topology preservation. Section 4.2

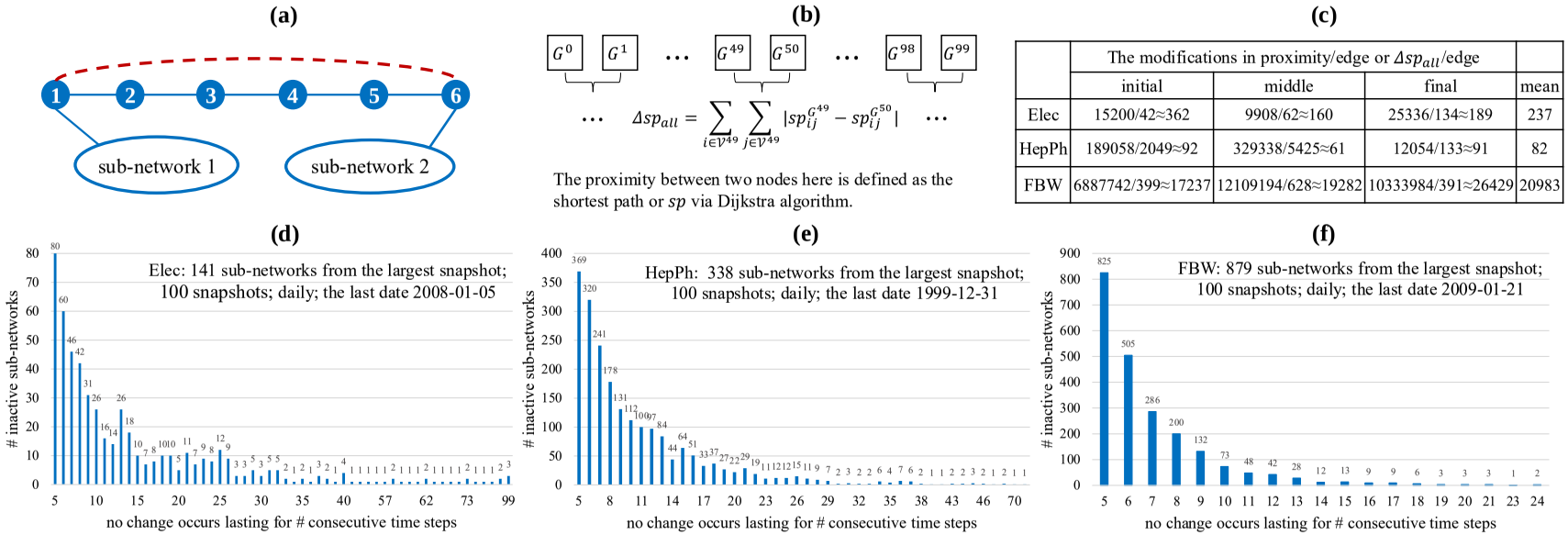
reviews the prior related work. In Section 4.3, we define the problem, present the proposed method, and analyze its complexity. The empirical studies are reported in Section 4.4, where Section 4.4.2 compares the proposed method to other methods and Section 4.4.3 investigates the proposed method itself. Finally, Section 4.5 summarizes this chapter.

4.1 Background

The interactions or connectivities between entities of a real-world complex system can be naturally represented as a network (or graph), e.g., social networks, biological networks, and sensor networks. Learning topological representation of a network, especially low-dimensional node embeddings which encode network topology therein so as to facilitate downstream tasks, has received a great success in the past few years [48, 49, 51].

Most previous network embedding methods such as [67, 68, 73, 74, 102] are designed for *static networks*. However, many real-world networks are dynamic by nature, i.e., edges might be added or deleted between seen and/or unseen nodes as time goes on. For instance, in a wireless sensor network, devices would regularly connect to or accidentally disconnect from routers; in a social network, new friendships would establish between new users and/or existing users. Due to the *time-evolving nature* of many real-world networks, Dynamic Network Embedding (DNE) is now attracting much attention [26–36]. The main and common objective of DNE is to efficiently update node embeddings while preserving network topology at each timestep. Most existing DNE methods try to compromise between effectiveness (evaluated by downstream tasks) and efficiency (while obtaining embeddings). The idea is to capture topological changes at or around the most affected nodes (instead of all nodes), and promptly update node embeddings based on an incremental learning paradigm.

Unfortunately, this kind of approximation, although can improve the efficiency, cannot effectively preserve the global topology of a dynamic network at each timestep. Specifically, any changes, i.e., edges being added or deleted, would affect all nodes in a connected network and greatly modify the proximity between nodes over a network via the high-order proximity as illustrated in Figure 4.1 a-c). On the other hand, as observed in Figure 4.1 d-f), the real-



	The modifications in proximity/edge or Δsp_{all}/edge			
	initial	middle	final	mean
Elec	15200/42≈362	9908/62≈160	25336/134≈189	237
HepPh	189058/2049≈92	329338/5425≈61	12054/133≈91	82
FBW	6887742/399≈17237	12109194/628≈19282	10333984/391≈26429	20983

Figure 4.1: a) A change (new edge in red) affects all nodes in the connected network via high-order proximity. The proximity of nodes 1-6 becomes 1st order from 5th order, nodes 2-6 becomes 2nd order from 4th order, etc. The proximity of any node in sub-network 1 to any node in sub-network 2 is reduced by 5 orders. b-c) How to calculate the modifications of the proximity between two snapshots, and the results show the modifications caused by a single edge can be very large in the real-world dynamic networks. d-f) The real-world dynamic networks have some inactive sub-networks, e.g., defined as no change occurs lasting for at least 5 timesteps. The sub-networks, each of which has about 50 nodes, are obtained by applying METIS algorithm [130] on the largest snapshot of a dynamic network. The details of the three dynamic networks are described in Section 4.4.1.1.

world dynamic networks usually have some *inactive sub-networks* where no change occurs lasting for several timesteps. Putting both together, the existing DNE methods that focus on the most affected nodes (belonging to the active sub-networks) but do not consider the inactive sub-networks, would overlook the accumulated topological changes propagating to the inactive sub-networks via the high-order proximity.

To tackle this challenge, the proposed DNE method, Global topology preserving Dynamic Network Embedding (GloDyNE), first partitions a current network into smaller sub-networks where one representative node in each sub-network is selected, so as to ensure the *diversity* of selected nodes. The representative node for each sub-network is sampled via a probability distribution over all nodes within each sub-network, such that a higher probability is assigned to a node with the larger accumulated topological changes. After that, GloDyNE captures the latest topologies around the selected nodes by truncated random walks [73], and then promptly updates node embeddings based on the Skip-Gram Negative Sampling (SGNS) model [69] and an incremental learning paradigm.

The contributions of this work are as follows.

- We demonstrate the existence of inactive sub-networks in real-world dynamic networks. Together with the propagation of topological changes via the high-order proximity, we find the issue of global topology preservation for many existing DNE methods.
- To better preserve the global topology, unlike all previous DNE methods, we propose to also consider the accumulated topological changes in the inactive sub-networks. A novel node selecting strategy is thus proposed to diversely select the representative nodes over a network.
- We further develop a new DNE method or framework, namely GloDyNE, which extends the random walk and Skip-Gram based network embedding approach to an incremental learning paradigm with a free hyper-parameter for controlling the number of selected nodes at each timestep.
- The empirical studies show the superiority of GloDyNE compared to the state-of-the-art DNE methods, as well as verify the usefulness of some special designs in GloDyNE such as the node selecting strategy and the free hyper-parameter.

4.2 Prior Related Work

To learn low-dimensional topological representation of a network in dynamic environments, one naive solution is to treat the snapshot of a dynamic network at each timestep as a static network, so that a static network embedding method such as [68, 73] can be directly applied to learn node embeddings for each snapshot. As reported in recent DNE works [26, 29–31], this naive solution obtains superior results compared with some DNE methods. One possible reason is that this solution does not suffer the aforementioned issue of global topology preservation. However, it is time-consuming [29, 31], and thus may not satisfy the requirement of promptly updating embeddings for some downstream tasks [33, 40].

To compromise between effectiveness and efficiency, most existing DNE methods try to capture the topological changes at or around the most affected nodes (instead of all nodes or edges), and promptly update node embeddings based on an incremental learning paradigm. BCGD [26] aims to minimize the loss of reconstructing the network proximity matrix using the node embedding matrix with a temporal regularization term, and it is optimized by the Block-Coordinate Gradient Descent algorithm. Particularly, this work further offers an efficient solution–BCGD-incremental that only updates the most affected nodes' embeddings based on their previous embeddings. DynGEM [28] and NetWalk [40] both utilize an auto-encoder with some regularization terms for modeling. They continuously train the model inherited from the last timestep, so that the model converges in a few iterations thanks to the knowledge transfer from previous models. To efficiently cope with dynamic changes at each timestep, some DNE methods [31, 34] propose an incremental version of the Skip-Gram model [69] to update embeddings based on the most affected nodes. Likewise, DHEP [29] extends HOPE [68] to an incremental version by modifying the most affected eigenvectors using matrix perturbation theory.

Apart from above DNE methods with the trade-off between effectiveness and efficiency, some DNE methods [32, 35, 36] aim to further improve effectiveness without considering the efficiency. For example, tNE [35] runs a static network embedding method to get node embeddings at each timestep, and then employs Recurrent Neural Networks among them

(for better exploiting the temporal dependence and hence may further improve effectiveness) to obtain the final node embeddings at each timestep.

Unlike the previous DNE works that mainly consider the most affected nodes or sub-networks, this work proposes to also consider the accumulated topological changes in inactive sub-networks to better preserve the global topology of a dynamic network. Moreover, this work focuses on the network topology, but does not include node attributes [14, 16, 18], which would be left as the future work.

4.3 Method

4.3.1 Problem Definition

Definition 1: A Static Network. Let $G = (\mathcal{V}, \mathcal{E})$ be a static network where $\mathcal{V}$ denotes a set of $|\mathcal{V}|$ nodes or vertices, and $\mathcal{E}$ denotes a set of $|\mathcal{E}|$ edges or links. The adjacency matrix of G is denoted as $\mathbf{W} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ where W_{ij} is the weight of edge e_{ij} between a pair of nodes (v_i, v_j). And if $W_{ij} = 0$, there is no edge between the two nodes.

Definition 2: A Dynamic Network. A dynamic network $\mathcal{G}$ is represented by a sequence of snapshots G^t taken at each timestep t, i.e., $\mathcal{G} = (G^0, G^1, ..., G^t, ...)$. Each snapshot $G^t(\mathcal{V}^t, \mathcal{E}^t)$ can be treated as a static network.

Definition 3: Static Network Embedding. Given a static network $G(\mathcal{V}, \mathcal{E})$, it aims to find a mapping $f : \mathcal{V} \mapsto \mathbf{Z}$ where $\mathbf{Z} \in \mathbb{R}^{|\mathcal{V}|\times d}$ is the output embedding matrix with a set of node embeddings; each row vector $\mathbf{Z}_i \in \mathbb{R}^d$ corresponds to node v_i in $\mathcal{V}$; and $d \ll |\mathcal{V}|$ is the user-specified embedding dimensionality. The objective is to best preserve some topologies of G while learning node embeddings.

Definition 4: Dynamic Network Embedding. Given a dynamic network $\mathcal{G} = \{G^0, ..., G^t\}$ with the latest snapshot $G^t(\mathcal{V}^t, \mathcal{E}^t)$, it aims to find a mapping $f^t : \mathcal{V}^t \mapsto \mathbf{Z}^t$ where $\mathbf{Z}^t \in \mathbb{R}^{|\mathcal{V}^t|\times d}$ is the output embedding matrix at timestep t; each row vector $\mathbf{Z}^t_i \in \mathbb{R}^d$ corresponds to node v_i in $\mathcal{V}^t$; and d is the user-specified embedding dimensionality. The main objective is to

efficiently update node embeddings at timestep t, so that the pairwise similarity of node embeddings in $\mathbf{Z}^t$ best preserves the pairwise topological similarity of nodes in G^t.

Definition 5: Sub-networks of A Snapshot. Let G_k^t denote the k-th sub-network in a snapshot G^t. All sub-networks of a snapshot G^t, after network partition [131], should be non-overlapping, i.e, $\mathcal{V}_m^t \cap \mathcal{V}_n^t = \emptyset$, $\forall \ \ m \neq n$. And their node sets should satisfy $\mathcal{V}^t = \bigcup_k \mathcal{V}_k^t$.

4.3.2 Method Description

The proposed DNE method GloDyNE consists of four essential components. Next, we introduce them step by step. Intuitively, Step 1 and 2 ensure the selected nodes *diversely distributed over a network*, and meanwhile, bias to the nodes with the larger accumulated topological changes in each sub-network. Step 3 encodes the latest topologies around the selected nodes into random walks (i.e., node sequences), which are then decoded by a sliding window and the SGNS model for incrementally training node embeddings as described in Step 4. Note that, these four steps are repeatedly executed at each timestep.

4.3.2.1 Step 1. Partition A Network

In order to realize inactivate sub-networks of a snapshot G^t, it is needed to divide G^t into sub-networks $G_1^t, G_2^t, ..., G_K^t$ where K is the number of sub-networks of a snapshot. The sub-networks are desirable to be non-overlapping and to cover all nodes in the original snapshot as defined in Definition 5, so that the later Step 2 can select unique nodes from each sub-network and the later Step 3 is easier to explore the whole snapshot G^t based on the selected nodes from each sub-network. A network partition algorithm [131] is therefore used to achieve the desirable goals. The most common objective function is to minimize the edge cut, i.e.,

$$\min \sum_{1 \leq m,n \leq K} \{W_{ij}^t \mid v_i^t \in \mathcal{V}_m^t, v_j^t \in \mathcal{V}_n^t, (v_i^t, v_j^t) \in \mathcal{E}^t\} \tag{4.1}$$

where the subscripts i, j indicate node ID and m, n indicate sub-network ID. Note that, Eq. (4.1) should subject to two constraints $\mathcal{V}_m^t \cap \mathcal{V}_n^t = \emptyset$, $\forall \ \ m \neq n$, and $\mathcal{V}^t = \bigcup_k \mathcal{V}_k^t$ for the reasons as discussed above.

Moreover, an additional constraint of the balanced sub-networks is introduced to let the number of nodes be similar among all sub-networks, so as to facilitate the later steps to fairly explore all sub-networks and hence better preserve the global topology. The third constraint about the balanced sub-networks can be defined as

$$\forall k \in \{1, ..., K\}, |\mathcal{V}_k^t| \leq (1 + \epsilon)\frac{|\mathcal{V}^t|}{K} \tag{4.2}$$

where $|\mathcal{V}_k^t|$ is the number of nodes in G_k^t and ϵ is the tolerance parameter. Note that, if ϵ is 0, network partitions are perfectly balanced. In practice, ϵ is set to a small number to allow a slight violation. However, such a (K, ϵ) balanced network partition is an NP-hard problem [131]. In order to address this problem, METIS algorithm [130] is employed. There are roughly three steps. First, the coarsening phase, the original network is recursively transformed into a series of smaller and smaller abstract networks, via collapsing nodes with common neighbors into one collapsed node until the abstract network is small enough. Second, the partition phase, a K-way partition algorithm is applied on the smallest abstract network to get the initial partition of K sub-networks. And third, the uncoarsening phase, it recursively expands the smallest abstract network back to the original network, and meanwhile recursively swaps the collapsed nodes (or the original nodes lastly) at the boarder of sub-networks between two neighboring sub-networks, so as to minimize the edge cut as describe in Eq. (4.1).

4.3.2.2 Step 2. Select Representative Nodes

In order to ensure the selected nodes diversely distributed over a snapshot G^t, one natural idea is to select one representative node from each sub-network. As a result, the total number of selected nodes is K. We let $K = \alpha|\mathcal{V}^t|$, so that α can freely control the total number of selected nodes for the trade-off between effectiveness and efficiency.

The problem now becomes as how to select one representative node from a sub-network. According to the latest DNE works such as [31, 34, 40], the nodes affected greatly by edge streams are selected for updating their embeddings, since their topologies are altered greatly. Similarly, in this work, the representative node to be selected is biased to the node with larger

topological changes. Motivated by the concept of inertia[1] from Physics, an efficient scoring function is designed to evaluate the accumulated topological changes of a node v_i^t in a current snapshot G^t as follows

$$S(v_i^t) = \frac{|\Delta\mathcal{E}_i^t| + \mathcal{R}_i^{t-1}}{\mathrm{Deg}(v_i^{t-1})} = \frac{|\,\mathcal{N}(v_i^t) \cup \mathcal{N}(v_i^{t-1}) - \mathcal{N}(v_i^t) \cap \mathcal{N}(v_i^{t-1})\,| + \mathcal{R}_i^{t-1}}{\mathrm{Deg}(v_i^{t-1})} \tag{4.3}$$

where the reservoir $\mathcal{R}_i^{t-1}$ stores the accumulated changes[2] of v_i up to $t-1$. For simplicity, we treat G^t as an undirected and unweighted network[3], so that the current changes of v_i at t, denoted as $|\Delta\mathcal{E}_i^t|$, can be easily obtained by the set operations on neighbors of v_i as shown in Eq. (4.3), which is equivalent to count the number of the edges with node v_i in the current edge streams $\Delta\mathcal{E}^t$. The representative node of a sub-network G_k^t is then selected based on the probability distribution over its node set $\mathcal{V}_k^t$, i.e.,

$$P(v_i^t) = \frac{e^{S(v_i^t)}}{\sum_{v_j^t \in \mathcal{V}_k^t} e^{S(v_j^t)}} \quad \forall\ v_i^t \in \mathcal{V}_k^t \tag{4.4}$$

where e is Euler's number and $S(v_i^t)$ is the score of the accumulated topological changes of node v_i^t given by Eq. (4.3). Note that, if $S(v_i^t) = 0$, $e^0 = 1$ and $P(v_i^t) \neq 0$, so that even for an inactivate sub-network with no change in all nodes, the probability distribution over this sub-network is still a valid uniform distribution. Intuitively, within a sub-network, the higher score of a node given by Eq. (4.3) is, the higher probability of this node would be selected as the representative node for this sub-network. Because one representative node from each sub-network is selected, all the selected nodes are therefore diversely distributed over the whole snapshot, and meanwhile, biased to the larger accumulated topological changes for each sub-network.

[1]Here the node degree is regarded as the inertia of this node.

[2]The accumulated changes in reservoir are used to handle the case when a node has small changes at each timestep for a long time, which greatly affects network topology but maybe ignored if not recorded.

[3]If one wants to consider edge's weight in Eq. (4.3), let $|\Delta\mathcal{E}_i^t| = \sum_{v_j^t \in \mathcal{N}(v_i^t)} |W_{ij}^t - W_{ij}^{t-1}| + \sum_{v_j^{t-1} \in (\mathcal{N}(v_i^{t-1}) - \mathcal{N}(v_i^t))} |W_{ij}^{t-1}|$ where the first term gives the total weight changes of i' neighbors presented at t; whereas the second term gives the total weight changes of i' neighbors presented at $t-1$ but not presented at t. The operator $|\cdot|$ on a set gives its cardinal number, and on a scalar gives its absolute value.

4.3.2.3 Step 3. Capture Topological Changes

Given the selected representative nodes from Step 2, this step explains how to capture the topological changes based on the selected nodes. As the topological changes at the selected nodes can propagate to other nodes via the high-order proximity, the truncated random walk sampling [73] (instead of edge sampling [102]) strategy is employed to capture the topological changes around (instead of at) the selected nodes. Concretely, for each selected node, r truncated random walks with length l are conducted starting from the selected node. For a random walk, the next node v_j^t is sampled based on the probability distribution over its previous node's neighbors $\mathcal{N}(v_i^t)$, i.e.,

$$P(v_j^t \mid v_i^t) = \begin{cases} \frac{W_{ij}^t}{\sum_{v_{j'} \in \mathcal{N}(v_i^t)} W_{ij'}^t} & \text{if } v_j \in \mathcal{N}(v_i^t) \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

4.3.2.4 Step 4. Update Node Embeddings

After Step 3, the latest topological information around the selected nodes is encoded in random walks. Step 4 aims to utilize the random walks to update node embeddings. Following [73] and [74], a sliding window with length $s + 1 + s$ is used to slide along each walk (i.e., node sequence), and the positive node-pair samples in D^t are built via $v_{center}^t, (v_{center+i}^t)$ where $i \in [-s, +s], i \neq 0$. As a result, the node-pair samples can encode $1^{st} \sim s^{th}$-order proximity of a given center node with another node. Note that, several network embedding works have shown the advantage of using the high-order proximity [67, 68, 132].

The latest topological information about changes captured in random walks has now been encoded into node co-occurrence statistics in D^t [121]. In addition, our aim is to learn node embeddings that reflect pairwise similarity between two nodes rather than to predict the probability of a random walk with l nodes [73]. Therefore, we could assume the observations of node pairs in D^t are mutually independent [74]. The objective function to maximize the node co-occurrence log probability over all node pairs in D^t can be written as

$$\max \sum_{(v_i^t, v_c^t) \in D^t} \log P(v_i^t \mid v_c^t) \tag{4.6}$$

where v_c^t is the center node, and v_i^t is another node with $1^{st} \sim s^{th}$ order proximity to v_c^t.

Unlike [73] that defines $P(v_i^t \mid v_c^t)$ using softmax, we treat it as a binary classification problem to reduce expensive calculations in the denominator of softmax. Concretely, the Skip-Gram Negative Sample (SGNS) model aims to distinguish a positive sample $(v_c^t, v_i^t) \in D^t$ from q negative samples $(v_c^t, v_{i'}^t)$s, which conceptually comes from noise contrastive estimation [133, 134], i.e., an efficient estimation of softmax by distinguishing observed data from some artificially generated noise using logistic regression [69]. The probability of observing a positive sample (v_i^t, v_j^t) is

$$P(B = 1 \mid v_c^t, v_i^t) = \sigma(\mathbf{Z}_c^t \cdot \mathbf{Z'}_i^t) = \frac{1}{1 + e^{-\mathbf{Z}_c^t \cdot \mathbf{Z'}_i^t}} \tag{4.7}$$

where $\mathbf{Z}_c^t$ and $\mathbf{Z'}_i^t$ are two vectors from two trainable matrices $\mathbf{Z}$ and $\mathbf{Z'}$ respectively; the operator $\cdot$ represents the dot product between two vectors; and $P(B = 1 \mid v_c^t, v_i^t)$ gives the probability of a positive prediction given a positive sample (v_c^t, v_i^t). Likewise, the probability of observing a negative sample $(v_c^t, v_{i'}^t)$ is

$$P(B = 0 \mid v_c^t, v_{i'}^t) = 1 - \sigma(\mathbf{Z}_c^t \cdot \mathbf{Z'}_{i'}^t) = \frac{1}{1 + e^{\mathbf{Z}_c^t \cdot \mathbf{Z'}_{i'}^t}} = \sigma(-\mathbf{Z}_c^t \cdot \mathbf{Z'}_{i'}^t) \tag{4.8}$$

where $P(B = 0 \mid v_c^t, v_{i'}^t)$ gives the probability of a negative prediction given a negative sample $(v_c^t, v_{i'}^t)$. Putting together, the objective of the SGNS model is to maximize $P(B = 1 \mid v_c^t, v_i^t)$ for each positive sample in D^t and $P(B = 0 \mid v_c^t, v_{i'}^t)$ for the q negative samples corresponding to each positive sample, i.e.,

$$\max \ \log \sigma(\mathbf{Z}_c^t \cdot \mathbf{Z'}_i^t) + \sum_{i'=1}^{q} \mathbb{E}_{v_{i'}^t \sim P_{D^t}} [\log \sigma(-\mathbf{Z}_c^t \cdot \mathbf{Z'}_{i'}^t)] \tag{4.9}$$

And the overall objective of the SGNS model is to sum over all positive samples and their negative samples, i.e.,

$$\max_{\mathbf{Z}^t, \mathbf{Z'}^t} \sum_{(v_c^t, v_i^t) \in D^t} \{\log \sigma(\mathbf{Z}_c^t \cdot \mathbf{Z'}_i^t) + \sum_{i'=1}^{q} \mathbb{E}_{v_{i'}^t \sim P_{D^t}} [\log \sigma(-\mathbf{Z}_c^t \cdot \mathbf{Z'}_{i'}^t)]\} \tag{4.10}$$

where $\mathbf{Z}_c^t \in \mathbb{R}^d$ is the embedding vector from row c of the trainable embedding matrix $\mathbf{Z}^t$ (between input and middle layers a.k.a. a project layer [69], and $\mathbf{Z}^t$ serves as DNE output after training), while $\mathbf{Z'}_i^t \in \mathbb{R}^d$ and $\mathbf{Z'}_{i'}^t \in \mathbb{R}^d$ are the embedding vector from column i or

i' of another trainable matrix $\mathbf{Z}'^t$ (between middle and output layers) [120]. Eq. (4.10) is optimized by stochastic gradient ascent over D^t to learn trainable embeddings. Intuitively, the more frequency two nodes appear in D^t, the closer their embeddings would be.

Finally, we extend SGNS as described above to an *incremental learning paradigm*. The overall framework of GloDyNE can be formalized as

$$\mathbf{Z}^t = \begin{cases} f^0(G^0, \mathbf{Z}^0_{\text{rand}}, \mathbf{Z}'^0_{\text{rand}}) & t = 0 \\ f^t(G^t, G^{t-1}, \mathbf{Z}^{t-1}, \mathbf{Z}'^{t-1}) & t \geq 1 \end{cases} \tag{4.11}$$

where f^t maps each node in G^t to corresponding node embeddings in $\mathbf{Z}^t$ using a neural network model in which $\mathbf{Z}^t$ is weights between input and middle layers, and $\mathbf{Z}'^t$ is weights between middle and output layers; the trained weights $\mathbf{Z}^{t-1}$ and $\mathbf{Z}'^{t-1}$ in f^{t-1} are copied to the trainable weights in f^t (denoted as $f^t \leftarrow f^{t-1}$) as the *initialization*; and G^{t-1} and G^t are two consecutive snapshots to provide dynamic changes. After f^t is trained by Eq. (4.10), we can directly take out $\mathbf{Z}^t$ as the output of DNE at timestep t.

4.3.3 Algorithm and Complexity

The pseudocode of GloDyNE is summarized in Algorithm 3, and the open source code is provided at `https://github.com/houchengbin/GloDyNE`

According to Eq. (4.11) and Algorithm 3, GloDyNE consists of two stages. During the *offline stage*, i.e., $t = 0$, Step 3 (specifically $\mathcal{V}^t_{\text{sel}} = \mathcal{V}^0_{\text{all}}$) and Step 4 are employed to obtain the initial SGNS model and node embeddings. Lines 2-5 are indeed a static network embedding method–a modified version of DeepWalk [73], which trains a SGNS model instead of Skip-Gram Hierarchical Softmax (SGHS) model. Therefore, the time complexity of lines 2-5 is further reduced to $O(rlw(1+q)d|\mathcal{V}^0_{\text{all}}|)$ [74] where $(1+q)$ is due to one positive sample corresponding to q negative samples.

During the *online stage*, i.e., $t \geq 1$, steps 1-4 are employed to incrementally update the SGNS model and node embeddings. For lines 7-8 corresponding to Step 1, the time complexity is $O(|\mathcal{V}^t| + |\mathcal{E}^t| + K \log K)$ [130] where $K = \alpha|\mathcal{V}^t|, \alpha \in (0, 1)$. For lines 9-14 corresponding to Step 2, the time complexity of lines 9-10 is $O(|\Delta\mathcal{E}^t|)$; the time complexity

Algorithm 3 GloDyNE: Global Topology Preserving Dynamic Network Embedding

Input: snapshots of a dynamic network $G^0, ..., G^{t-1}, G^t$,...; previous trained model f^{t-1}; coefficient to determine the number of selected nodes α; walks per node r; walk length l; sliding window size s; negative samples per positive sample q; embedding dimensionality d

Output: embedding matrix $\mathbf{Z}^t \in \mathbb{R}^{|\mathcal{V}^t| \times d}$ at each timestep

for $t = 0$ **do**
  conduct random walks with length l starting from each node in $\mathcal{V}^0_{\text{all}}$ for r times by Eq. (4.5)
  build positive samples D^0 based on each sliding window with size s along each walk
  initialize SGNS f^0_{rand}, and train it using D^0 each with q negative samples by Eq. (4.10)
  return f^0 and $\mathbf{Z}^0$
for $t \geq 1$ **do**
  calculate $K = \alpha|\mathcal{V}^t|$, and initialize $\Delta\mathcal{E}^t = \emptyset$ and $\mathcal{V}^t_{\text{sel}} = \emptyset$
  partition G^t into K sub-networks $G^t_1, G^t_2, ..., G^t_K$ by METIS based on Eq. (4.1) and (4.2)
  read edge streams $\Delta\mathcal{E}^t$ (or obtain it by differences between G^{t-1} and G^t if not given)
  update reservoir via $\mathcal{R}^t_{v_i} = |\Delta\mathcal{E}^t_i| + \mathcal{R}^{t-1}_i$ for accumulating new changes of v^t_i by Eq. (4.3)
  for $k \in \{1, ..., K\}$ **do**
    calculate a probability distribution over all nodes in a sub-network G^t_k by Eq. (4.4)
    select one representative node based on the probability distribution, and add it to $\mathcal{V}^t_{\text{sel}}$
  remove selected nodes $\mathcal{V}^t_{\text{sel}}$ from the reservoir $\mathcal{R}^t$ if exists
  conduct random walks with length l starting from each node in $\mathcal{V}^t_{\text{sel}}$ for r times by Eq. (4.5)
  build positive samples D^t based on each sliding window with size s along each walk
  initialize SGNS $f^t \leftarrow f^{t-1}$, and train it using D^t each with q negative samples by Eq. (4.10)
  return f^t and $\mathbf{Z}^t$

of lines 11-13 using alias sampling method [74] requires $O(|\mathcal{V}^t|)$; the time complexity of line 14 is $O(\alpha|\mathcal{V}^t|)$ due to $|\mathcal{V}^t_{\text{sel}}| = K = \alpha|\mathcal{V}^t|$. For lines 14-18 corresponding to Step 3 and Step 4, similarly to lines 2-5 above, the complexity of lines 14-18 is $O(rlw(1+q)d|\mathcal{V}^t_{\text{sel}}|)$ where $|\mathcal{V}^t_{\text{sel}}| = \alpha|\mathcal{V}^t|$. Because most real-world networks are sparse, edges in a snapshot $|\mathcal{E}^t| = b_1|\mathcal{V}^t|$ such that the average degree b_1 is a very small number compared with $|\mathcal{V}^t|$. Besides, since the edge streams between two consecutive snapshots are often much less than the edges in the snapshot, edge streams $|\Delta\mathcal{E}^t| = b_2|\mathcal{V}^t|$ such that $b_2 < b_1 \ll |\mathcal{V}^t|$. Regarding those real-world assumptions, the overall complexity of online stage (including all four steps) at each timestep can be approximated as $O(|\mathcal{V}^t| + \alpha|\mathcal{V}^t| \log\ \alpha|\mathcal{V}^t| + rlw(1+q)d\alpha|\mathcal{V}^t|)$ where $\alpha \in (0, 1)$ is used to control the number of selected nodes; $|\mathcal{V}^t|$ denotes the number of nodes at t; and others are negligible constants compared to $|\mathcal{V}^t|$. GloDyNE is thus scalable w.r.t. $|\mathcal{V}^t|$, as there is no quadratic or higher term.

4.4 Experiments

4.4.1 Experimental Settings

4.4.1.1 Datasets

In this work, six datasets are employed to evaluate the proposed method.

- AS733 contains 733 daily instances of the autonomous system of routers exchanging traffic flows with neighbors. Since AS733 is directly given as the snapshot representation, we directly take out the recent 21 snapshots (12/Dec./1991–01/Jan./2000) to form its dynamic network. The initial snapshot has 1476 nodes and 3123 edges, and the final snapshot has 3570 nodes and 7033 edges. The original dataset comes from `https://snap.stanford.edu/data/as-733.html`
- Elec is the network of English Wikipedia users vote for and against each other in admin elections. The gap between the timestamps for taking snapshots is one calendar day. We take out the recent 21 snapshots (16/Dec./2007–05/Jan./2008) to form its dynamic network. The initial snapshot has 6972 nodes and 99006 edges, and the

final snapshot has 7066 nodes and 100655 edges. The original dataset comes from `http://konect.cc/networks/elec`

- FBW is a social network of Facebook Wall posts where nodes are users and edges are built based on the interactions in wall posts. The gap between the timestamps for taking snapshots is one calendar day. We take out the recent 21 snapshots (01/Jan./2009–21/Jan./2009) to form its dynamic network. The initial snapshot has 41730 nodes and 169918 edges, and the final snapshot has 43952 nodes and 182365 edges. The original dataset comes from `http://konect.cc/networks/facebook-wosn-wall`
- HepPh is a co-author network extracted from the papers of High Energy Physics Phenomenology in arXiv. The gap between the timestamps for taking snapshots is one month. We take out the recent 21 snapshots (Apr./1998–Dec./1999) to form its dynamic network. The initial snapshot has 11156 nodes and 611311 edges, and the final snapshot has 16913 nodes and 1194408 edges. The original dataset comes from `http://konect.cc/networks/ca-cit-HepPh`
- Cora is a citation network where each node represents a paper, and an edge between two nodes represents a citation. Each paper is assigned with a label (from 10 different labels) based on its field of the publication. Following [35], the gap between the timestamps for taking snapshots is one year. The 11 snapshots (1989–1999) are taken out to form its dynamic network. The initial snapshot has 348 nodes and 481 edges, and the final snapshot has 12022 nodes and 45421 edges. The original dataset comes from `https://people.cs.umass.edu/~mccallum/data.html`
- DBLP is a co-author network in computer science field. Each author is associated with a label (from 15 different labels). The label of an author is defined by the fields in which the author has the most publications. Following [35], the gap between the timestamps for taking snapshots is one year. The 11 snapshots (1985–1995) are taken out to form its dynamic network. The initial snapshot has 1679 nodes and 3445 edges, and the final snapshot has 25826 nodes and 56932 edges. The original dataset comes from `https://dblp.org/xml/release`

To construct the dynamic networks, except AS733 (given as the snapshot representation),

all other ones (given as the edge streams $\{(v_i, v_j, timestamp), ...\}$) are constructed as follows: 1) the initial snapshot G^0 is built by appending all edges no later than the initial cut-off timestamp; 2) the next snapshot G^1 is built by appending the edges newly appeared until the next cut-off timestamp to G^0; 3) repeat step 2 so as to generate G^2, G^3, and so on. For each snapshot, we take out the largest connected component and treat it as an undirected and unweighted graph. According to real-world practice, the cut-off timestamp is based on the last second of a calendar day, and the gap between snapshots on a same dataset is identical. The gap for different datasets would be different due to the nature of different datasets, e.g., the gap is set to more calendar days if a dataset evolves more slowly over time.

4.4.1.2 Compared Methods

The proposed DNE method GloDyNE is compared with the following state-of-the-art DNE methods for demonstrating the effectiveness and efficiency.

- BCGD^g [26]: The general objective of BCGD is to minimize the quadratic loss of reconstructing the network proximity matrix using the node embedding matrix with a temporal regularization term. BCGD^g (or BCGD-global) employs all historical snapshots to jointly and cyclically update embeddings for all timesteps.
- BCGD^l [26]: Unlike BCGD^g but following the same general objective of BCGD as above, BCGD^l (or BCGD-local) iteratively employs the previous snapshot and initializes current embeddings with the previous embeddings to update embeddings for a current timestep.
- DynGEM [28]: This work proposes a strategy to modify the structure of a deep auto-encoder model based on the size of a current snapshot. At each timestep, the auto-encoder model is initialized by its previous model. DynGEM continuously trains the adaptive auto-encoder model based on the existing edges in a current snapshot.
- DynLINE [31]: This work extends the static network embedding method–LINE[102] to cope with dynamic networks. To improve efficiency, DynLINE updates the embeddings for the most affected nodes and new nodes in each snapshot.

- DynTriad [32]: DynTriad models the triadic closure process, the social homophily, and the temporal smoothness in its objective function to learn node embeddings at each timestep. It optimizes the objective function according to the existing edges of each snapshot respectively.
- tNE [35]: tNE runs a static network embedding method to get node embeddings for each snapshot, and then exploits the temporal dependence among all available static node embeddings using Recurrent Neural Networks, so as to obtain the final node embeddings for a current timestep.

The original open source codes with the default settings of BCGD[4], DynGEM[5], DynLINE[6], DynTriad[7], and tNE[8] are adopted in the experiments. Note that, BCGD^g and BCGD^l are two proposed algorithms in BCGD correspondingly to the type of algorithm 2 and 4. Moreover, we adopt the link prediction architecture of tNE to obtain node embeddings, so that all methods only use network linkage information as the supervised signal to learn node embeddings. Furthermore, the dimensionality of node embeddings is set to 128 for all methods for the fair comparison.

Regarding our method GloDyNE, following [73] and [74], the hyper-parameters of walks per node, walk length, window size, and negative samples are set to 10, 80, 10, and 5 respectively. The hyper-parameter α to control the number of selected nodes for freely trade-off between effectiveness and efficiency, is set to 0.1 unless otherwise specified.

4.4.2 Results and Discussions, Comparative Study

In this section, we conduct a comparative study over different methods. Three typical types of downstream tasks are employed to evaluate the quality of obtained node embeddings by the seven methods on the six datasets. Specifically, the *graph reconstruction* (GR) task is used to demonstrate the ability of global topology preservation, while the *link prediction*

[4] `https://github.com/linhongseba/Temporal-Network-Embedding`
[5] `https://github.com/palash1992/DynamicGEM`
[6] `https://github.com/lundu28/DynamicNetworkEmbedding`
[7] `https://github.com/luckiezhou/DynamicTriad`
[8] `https://github.com/urielsinger/tNodeEmbed`

(LP) task and *node classification* (NC) task are used to show the benefit of global topology preservation. For fairness, we first take out the node embeddings obtained by each method respectively, and then feed them to exactly the same downstream tasks. The above process is repeated for 20 runs. Their average results as well as other statistics are reported in Section 4.4.2.1, 4.4.2.2, and 4.4.2.3. Moreover, the average results of the wall-clock time to obtain node embeddings, are reported in Section 4.4.2.4 for comparing the efficiency of the implementation of the seven methods. Finally, the overall performance regarding both effectiveness and efficiency is discussed in Section 4.4.2.5.

All experiments in this section are conducted in the following hardware specification. For all methods, we enable 32 Intel-Xeon-E5-2.2GHz logical CPUs and 512G memory. In addition, for DynGEM, DynTriad, DynLINE and tNE that can use GPU for acceleration, we also enable 1 Nvidia-Tesla-P100 GPU with 16G memory. The n/a values for DynLINE and tNE on AS733 are due to the inability of handling node deletions. The n/a values for DynGEM on HepPh and FBW are because of running out of GPU memory.

Note that, only Cora and DBLP have node labels. During the embedding phase, there is no testing set, but the learned node embeddings (not dedicated to a specific downstream task) are evaluated by various downstream tasks. For the downstream tasks, there might be a training and/or testing set depending on the nature of a downstream task.

4.4.2.1 Graph Reconstruction (GR)

In order to demonstrate the ability of the global topology preservation of each method, one possible way is to use the obtained node embeddings to reconstruct the original network. For this purpose, precision at k or $P@k$ is used as the metric to evaluate how well the top-k similar nodes of each node in the embedding space can match the ground-truth neighbors of each node in the original network [29, 67, 132]. Concretely, $P@k(v_i) = |\mathcal{Q}(v_i)_{@k} \cap \mathcal{N}(v_i)| \ / \ \min(k, |\mathcal{N}(v_i)|)$, where $\mathcal{Q}(v_i)_{@k}$ gives a set of the top-k similar nodes of a queried node v_i based on the cosine similarity between node embeddings, and $\mathcal{N}(v_i)$ denotes the ground-truth neighbors of v_i. As a result, the testing set is just a node set to query, but there is no training set. To show the ability of global topology preserva-

Table 4.1: Comparative study for GR tasks by Mean$P$@k scores (in %). Each entry is obtained by the mean of Mean$P$@k over all timesteps, and then the mean with its standard deviation over 20 runs. Student's T-Test is applied to the best two results (in bold). The best result is indicated by † or ‡, if the p-value is < 0.05 or < 0.01. The n/a values are due to the inability of handling node deletions or running out of memory.

	AS733	Cora	DBLP	Elec	FBW	HepPh
			Mean$P$@1			
BCGD^g	02.13±0.06	02.09±0.16	02.17±0.08	10.01±0.05	00.32±0.01	32.81±0.07
BCGD^l	38.47±1.33	06.31±0.26	04.54±0.28	25.77±0.73	07.09±0.11	73.25±0.37
DynGEM	00.89±0.03	11.26±0.55	30.47±0.60	04.41±0.06	n/a	n/a
DynLINE	n/a	06.77±0.00	21.49±0.00	01.76±0.00	00.45±0.00	51.24±0.00
DynTriad	**65.43±18.76**	**64.68±23.85**	68.91±21.42	**69.86±8.48**‡	**76.92±7.55**	**80.64±3.67**
tNE	n/a	62.58±0.27	**72.20±0.09**	08.27±0.09	40.14±0.39	73.58±0.46
GloDyNE	**66.47±0.33**	**77.41±0.27**†	**81.85±0.11**‡	**51.75±0.16**	**90.63±0.04**‡	**84.21±0.13**‡
			Mean$P$@5			
BCGD^g	02.00±0.05	10.23±0.32	00.68±0.03	10.41±0.04	00.15±0.00	31.08±0.04
BCGD^l	42.61±2.07	05.75±0.84	02.67±0.15	20.64±0.63	05.69±0.13	65.91±0.39
DynGEM	00.82±0.02	07.22±0.36	22.62±0.48	04.14±0.04	n/a	n/a
DynLINE	n/a	09.45±0.00	26.58±0.00	01.54±0.00	00.28±0.00	44.36±0.00
DynTriad	**62.18±17.47**	53.40±23.13	56.15±22.02	**66.64±9.13**	**58.84±11.09**	**73.96±4.35**
tNE	n/a	**60.81±0.34**	**68.16±0.19**	06.82±0.06	27.54±0.34	61.24±0.44
GloDyNE	**70.37±0.26**†	**79.66±0.08**‡	**84.24±0.12**‡	**66.42±0.11**	**87.81±0.02**‡	**76.24±0.11**†
			Mean$P$@10			
BCGD^g	03.37±0.09	18.60±0.24	01.50±0.14	10.28±0.05	0.12±0.00	30.82±0.04
BCGD^l	51.94±2.82	08.04±2.70	03.30±0.16	19.22±0.64	4.67±0.09	61.12±0.38
DynGEM	00.82±0.02	07.28±0.33	22.54±0.50	03.99±0.03	n/a	n/a
DynLINE	n/a	12.80±0.00	33.76±0.00	01.39±0.00	00.23±0.00	40.34±0.00
DynTriad	**67.19±16.79**	55.05±23.22	57.60±21.94	**67.37±8.95**	**56.38±11.29**	**70.31±4.73**
tNE	n/a	**67.98±0.38**	**77.53±0.24**	06.37±0.05	27.05±0.35	55.16±0.42
GloDyNE	**78.25±0.20**‡	**86.53±0.11**‡	**94.01±0.07**‡	**71.19±0.09**	**89.22±0.01**‡	**72.43±0.11**
			Mean$P$@20			
BCGD^g	50.34±0.71	26.88±0.18	13.08±0.32	09.63±0.05	00.11±0.00	30.27±0.03
BCGD^l	69.62±4.34	14.21±4.00	12.97±0.2	19.44±0.71	03.96±0.06	56.76±0.36
DynGEM	00.89±0.02	08.52±0.36	24.33±0.51	03.81±0.03	n/a	n/a
DynLINE	n/a	17.53±0.00	39.39±0.00	01.30±0.00	00.20±0.00	36.85±0.00
DynTriad	**73.00±15.66**	59.54±23.47	61.66±21.98	**69.47±8.60**	**57.97±11.40**	**67.12±5.16**
tNE	n/a	**76.44±0.42**	**85.15±0.27**	06.17±0.06	29.03±0.37	49.48±0.41
GloDyNE	**85.40±0.17**†	**93.15±0.02**‡	**98.84±0.02**‡	**74.31±0.07**†	**91.79±0.01**‡	**69.91±0.10**†
			Mean$P$@40			
BCGD^g	**89.60±0.74**	36.96±0.16	35.89±0.29	08.85±0.05	00.12±0.00	29.06±0.03
BCGD^l	84.52±5.27	23.57±4.48	28.44±0.35	27.57±0.95	04.16±0.11	54.25±0.39
DynGEM	01.20±0.05	10.61±0.42	26.77±0.53	03.67±0.04	n/a	n/a
DynLINE	n/a	21.81±0.00	43.86±0.00	01.39±0.00	00.22±0.00	33.89±0.00
DynTriad	78.90±14.01	64.95±23.12	66.33±21.67	**72.80±8.03**	**61.92±11.53**	**65.42±5.61**
tNE	n/a	**81.27±0.41**	**88.57±0.30**	06.52±0.07	32.61±0.40	44.58±0.38
GloDyNE	**90.87±0.13**‡	**95.31±0.01**‡	**99.85±0.00**‡	**76.95±0.04**†	**94.95±0.00**‡	**69.81±0.08**‡

tion, we further calculate the mean of $P@k$ over all nodes in a current snapshot G^t, i.e., Mean$P@k$ = [$\sum_{v_i^t \in \mathcal{V}^t} P@k(v_i^t)$] / $|\mathcal{V}^t|$ where $\mathcal{V}^t$ is a set of all nodes in G^t, and $|\mathcal{V}^t|$ counts the number of nodes in $\mathcal{V}^t$. Table 4.1 presents the results for Mean$P@1$, Mean$P@5$, Mean$P@10$, Mean$P@20$, and Mean$P@40$.

First, GloDyNE consistently outperforms all other methods on all datasets (28/30 cases), except that DynTriad outperforms GloDyNE on Elec dataset under Mean$P@1$ and Mean$P@5$ (2/30 cases). Second, although DynTriad can obtain the second best results in many cases, the standard deviation of DynTriad is always very high (often larger than 5.0%). In contrast, the performance of GloDyNE is very stable, since the standard deviation is always very low (often smaller than 0.3%). Third, GloDyNE significantly outperforms the second best method in 25/30 cases according to the statistical hypothesis testing[9].

The main reason of such superiority of GloDyNE in the GR task is that GloDyNE is designed to better preserve the global topology of a dynamic network at each timestep, while the GR task is also employed for demonstrating the ability of global topology preservation.

4.4.2.2 Link Prediction (LP)

The (dynamic) LP task aims to predict future edges at timestep $t + 1$ using the obtained node embeddings at t. The testing edges include both added and deleted edges from t to $t + 1$, plus other edges randomly sampled from the snapshot at $t + 1$ for balancing existent edges (or positive samples) and non-existent edges (or negative samples). The LP task is then evaluated by Area Under the ROC Curve (AUC) score based on the cosine similarity between node embeddings [17, 26, 64]. Table 4.2 presents the AUC scores for LP tasks, and each entry of the table is obtained in the similar way as described in Table 4.1.

According to the statistical hypothesis testing, GloDyNE significantly outperforms the second best method on AS733, Cora, and FBW by 12.64%, 11.97% and 4.03% respectively (3/6 cases), while GloDyNE obtains the second best results on other three datasets (3/6 cases). Overall, GloDyNE is also a good method for the (dynamic) LP task on most datasets,

[9]Two-tailed and two-sample Student's T-Test is applied with the null hypothesis that there is no statistically significant difference of the mean over 20 runs between the two best results.

Table 4.2: Comparative study for (dynamic) LP tasks by AUC scores.

	AS733	Cora	DBLP	Elec	FBW	HepPh
BCGD^g	**69.64±0.64**	67.92±0.95	66.55±0.85	81.22±0.91	82.90±0.29	77.06±0.15
BCGD^l	62.69±1.05	**81.46±1.64**	**84.86±1.17**‡	86.61±3.72	**83.83±1.34**	**88.16±1.93**‡
DynGEM	61.60±1.28	56.69±1.42	60.90±0.81	60.71±1.56	n/a	n/a
DynLINE	n/a	65.20±0.85	57.21±0.58	58.80±0.68	65.14±0.31	62.20±0.03
DynTriad	64.36±3.45	65.86±6.37	58.78±2.66	**94.25±1.50**‡	78.54±4.81	85.10±2.63
tNE	n/a	79.94±0.68	59.26±0.71	61.50±1.19	68.38±0.48	77.16±0.52
GloDyNE	**82.10±0.32**‡	**93.43±0.36**‡	**74.56±0.67**	**87.03±0.85**	**87.86±0.18**‡	**85.88±0.05**

thanks to the high-order proximities being used for better preserving the global topology [67]. In fact, the high-order proximity between nodes is an important temporal feature for predicting future edges. For example, the triadic closure process which tries to predict the third edge among three nodes if there have already been two edges among them, as modelled in DynTriad [32], can be easily realized by considering the second-order proximity via setting $l \geq 3$ and $s \geq 3$ (see Section 4.3.2.3 and 4.3.2.4). In the experiments, we set $l = 80$ and $s = 10$. As a result, much higher order proximities (up to 10^{th} order according to s) are considered for better preserving the global topology, which therefore provides more advanced temporal features (analogous to triadic closure process) to improve the performance of GloDyNE in LP tasks on most datasets.

However, not all high-order proximities are helpful on all kinds of datasets. For example, we can observe from Table 4.1 and 4.2 that DynTriad, which mainly considers the second-order proximity due to modeling the triadic closure process, can obtain the best performance on Elec dataset; while GloDyNE, which considers more high-order proximities, obtains the second best performance. This observation might be caused by some special characteristics of Elec, which is an election network and is quite different from other kinds of networks.

4.4.2.3 Node Classification (NC)

The NC task aims to infer the most likely label for the nodes without labels. Specifically, 50%, 70%, and 90% nodes are randomly picked respectively to train a one-vs-rest logistic regression classifier based on their embeddings and labels. The left nodes respectively are

treated as the testing set. Note that, only Cora and DBLP are employed in NC tasks, as other datasets do not have node labels. At each timestep, the latest node embeddings are employed as the input features to logistic regression classifier. The prediction of the trained classifier over the testing set are evaluated by Micro-F1 and Macro-F1 [35, 73, 74] respectively. Table 4.3 presents the F1 scores for NC tasks, and each entry of the table is obtained in the similar way as described in Table 4.1.

Table 4.3: Comparative study for NC tasks by Micro-F1 and Macro-F1 scores. Three different proportions of training set are evaluated respectively.

	Cora			DBLP		
	0.5	0.7	0.9	0.5	0.7	0.9
			Micro-F1			
BCGD^g	32.28±0.29	33.43±0.49	33.93±0.74	56.00±0.17	56.15±0.15	56.34±0.16
BCGD^l	36.49±0.53	38.24±0.66	39.59±0.72	56.48±0.18	56.71±0.27	57.12±0.35
DynGEM	36.92±0.69	38.95±0.60	41.03±0.85	55.22±0.04	55.37±0.08	55.76±0.13
DynLINE	40.48±0.00	42.21±0.00	42.83±0.00	55.95±0.00	56.77±0.00	57.46±0.00
DynTriad	36.54±3.61	37.67±3.74	38.57±3.70	55.60±0.55	56.22±0.46	56.86±0.65
tNE	**65.37±0.29**	**67.12±0.29**	**67.97±0.49**	**63.19±0.16**	**63.90±0.20**	**64.27±0.36**
GloDyNE	**74.20±0.30**‡	**75.22±0.39**‡	**75.54±0.58**‡	**64.73±0.28**‡	**65.17±0.30**‡	**66.40±0.37**‡
			Macro-F1			
BCGD^g	08.30±0.20	08.68±0.20	08.58±0.38	10.24±0.14	10.26±0.20	10.19±0.31
BCGD^l	12.23±0.62	12.85±0.79	13.32±0.86	11.19±0.27	11.44±0.35	11.56±0.39
DynGEM	09.91±0.56	10.79±0.50	11.33±0.63	08.25±0.06	08.41±0.11	08.63±0.16
DynLINE	22.65±0.00	23.98±0.00	23.91±0.00	16.32±0.00	16.94±0.00	15.74±0.00
DynTriad	16.92±4.01	17.71±4.30	17.64±4.37	12.81±0.74	13.06±0.81	13.05±1.10
tNE	**50.03±0.31**	**52.45±0.39**	**51.41±0.96**	**25.60±0.40**	**27.02±0.35**	**26.02±0.84**
GloDyNE	**61.20±0.66**‡	**62.75±0.71**‡	**62.01±1.00**‡	**29.87±0.75**‡	**30.28±0.81**‡	**29.99±1.05**‡

According to the statistical hypothesis testing, GloDyNE significantly outperforms the second best method on both detests, which demonstrates the benefit of global topology preservation in NC tasks. Moreover, GloDyNE achieves better performance on Cora than DBLP. The reason is that Cora is a citation network where the label/field of nodes/papers contains less noise (the field of a journal or conference often remains the same), while DBLP is a co-author network where the label/field of nodes/authors contains more noise (the field of an author varies over time or an author with few papers is not accurate).

4.4.2.4 Walk-Clock Time During Embedding

To conduct the downstream tasks in Section 4.4.2.1, 4.4.2.2, and 4.4.2.3, the common step is to first obtain node embeddings which serve as the low dimensional hidden features to the downstream tasks. In this section, the wall-clock time of obtaining node embeddings (but not including downstream tasks) over all timesteps are reported in Table 4.4.

Table 4.4: Comparative study for wall-clock time (while obtaining node embeddings but not including downstream tasks) in seconds. Each result is given by the sum of the wall-clock time over all timesteps, and then the mean over 20 runs. The total number of nodes and edges of a dynamic network over all snapshots is also attached.

	AS733	Cora	DBLP	Elec	FBW	HepPh
BCGD^g	2987	4486	9277	6513	30063	24119
BCGD^l	597	1214	2543	1941	10272	12091
DynGEM	2021	3336	8054	1577	n/a	n/a
DynLINE	n/a	809	814	1577	1740	2095
DynTriad	109	208	318	1879	3408	16893
tNE	n/a	2728	4497	9989	79987	66679
GloDyNE	**64**	**106**	**186**	**203**	**943**	**908**
# of nodes	45 k	66 k	108 k	147 k	902 k	295 k
# of edges	91 k	216 k	233 k	2095 k	3703 k	18491 k

It is obvious that GloDyNE is the most efficient method among all methods on all datasets. In addition, the superiority of *efficiency* of GloDyNE grows, as the size of a dynamic network (given by the number of nodes or edges over all snapshots) grows. The reasons are as follows. First, GloDyNE is scalable regarding its time complexity, since there is no quadratic or higher term in $|\mathcal{V}|$ and $|\mathcal{E}|$ as analyzed in Section 4.3.3. Second, the implementation of Step 4 of GloDyNE is highly parallelized and optimized.

To further test the *scalability* of GloDyNE on a very large-scale dataset, a hyperlink network from `http://konect.cc/networks/link-dynamic-dewiki` is employed. We follow the same approach as described in Section 4.4.1.1 to generate its dynamic network. Specifically, the gap between snapshots is one calendar day; the recent 11 snapshots (03/Aug./2011-13/Aug./2011) are taken out; the initial snapshot has 2,161,514 nodes and 39,578,432 edges;

the final snapshot has 2,165,677 nodes and 39,705,237 edges; and the total number of nodes and edges over all snapshots are 23,795,061 and 435,918,113. During the embedding phase, for the initial snapshot or timestep (i.e., offline stage), the wall-clock time for Step 3 and Step 4 is 110698s and 12258s. After that, the averaged wall-clock time *per snapshot* over other 10 snapshots (i.e., online stage) for Step 1-2, Step 3, and Step 4 are 2769s (for network partition and node selection), 12388s (for capturing topological changes), and 1255s (for updating node embeddings) respectively. We may ignore the offline stage as it is only conducted once at the beginning. During the online stage, the overall time to obtain embeddings for one snapshot is 16412s or 4.56h, which is acceptable for the scenario of daily updating embeddings in this very large-scale hyperlink network. Note that, one may further reduce the overall time by parallelizing random walks over multiprocessors in Step 3, so as to overcome the main bottleneck of the current implementation of GloDyNE.

4.4.2.5 Effectiveness and Efficiency

To better visualize the comparison among the seven methods in terms of both effectiveness and efficiency, we make scatter plots as shown in Figure 4.2 based on the quantitative results in above sections. Note that, the wall-clock time (of a method on a dataset) to obtain node embeddings in Section 4.4.2.4 is the same for different downstream tasks. Besides, the n/a values in the tables are omitted in Figure 4.2.

In terms of both effectiveness and efficiency, GloDyNE is the best choice on AS733, Cora and FBW, since it is located at the most top-left corner in these cases. Although GloDyNE obtains the second best effectiveness on other three datasets, it still keeps the best efficiency in these cases. According to the quantitative results in Table 4.2 and Table 4.4, from the *effectiveness* perspective, GloDyNE is outperformed by BCGD^l, DynTriad and BCGD^l by 10.3%, 7,22% and 2.28% on DBLP, Elec and HepPh respectively. Nevertheless, from the *efficiency* perspective, GloDyNE is $\times$11.5, $\times$9.3 and $\times$13.3 faster respectively. Therefore, if one prefers efficiency, GloDyNE would be a better choice than BCGD^l, DynTriad and BCGD^l on DBLP, Elec and HepPh respectively.

Apart from the above discussions for LP tasks based on Figure 4.2, the scatter plots of

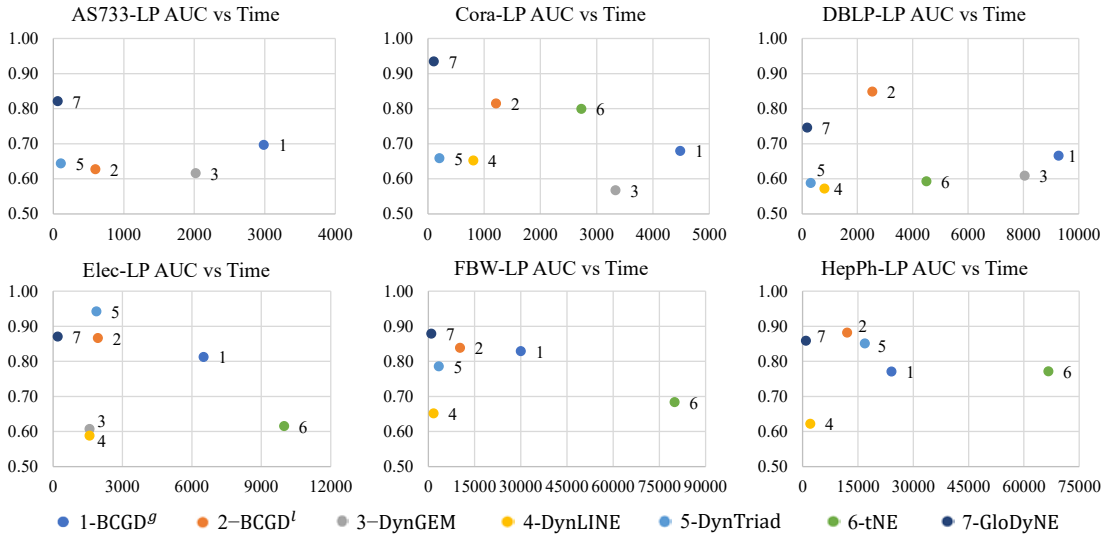


Figure 4.2: The comparison among the seven methods in terms of both effectiveness (y-axis for scores in decimal) and efficiency (x-axis for wall-clock time in seconds). We only show the plots of LP tasks for illustration. But the plots of GR and NC tasks are omitted, because GloDyNE obtains both the best effectiveness and best efficiency (i.e., located at the most top-left corner or the best choice) in almost all cases.

GR and NC tasks are omitted. The reason is that GloDyNE is the best choice in terms of both effectiveness and efficiency (i.e., located at the most top-left corner) for 28/30 cases of GR tasks and 12/12 cases of NC tasks.

4.4.3 Results and Discussions, Further Investigation

In this section, we further investigate the proposed method itself. Since GloDyNE is proposed to better preserve the global topology, we focus on the ability of global topology preservation, and thus adopt the *graph reconstruction* task for discussions. Besides, thanks to the good time and space efficiency of GloDyNE and its variants, all experiments in this section are conducted with less expensive hardware: 16 Intel-Xeon-E5-2.2GHz logical CPUs and 8G memory. All experiments are repeated for 20 runs. We choose two datasets for illustration. One is AS733 that includes both node additions and deletions, while another is Elec that only includes node additions.

4.4.3.1 Necessity of Dynamic Network Embedding

One advantage of DNE is that, it promptly updates node embeddings at each timestep, so that the latest embeddings can better reflect the original network topology at each timestep. To demonstrate this, two variants of GloDyNE based on the SGNS model, namely SGNS-static and SGNS-retrain, are used for comparison. For SGNS-static, we perform the $t = 0$ part of Algorithm 3, and the obtained embeddings at $t = 0$ are identically used in the downstream task at each timestep. For SGNS-retrain, we repeatedly perform the $t = 0$ part of Algorithm 3 at each timestep, and the obtain embeddings at each timestep are used in

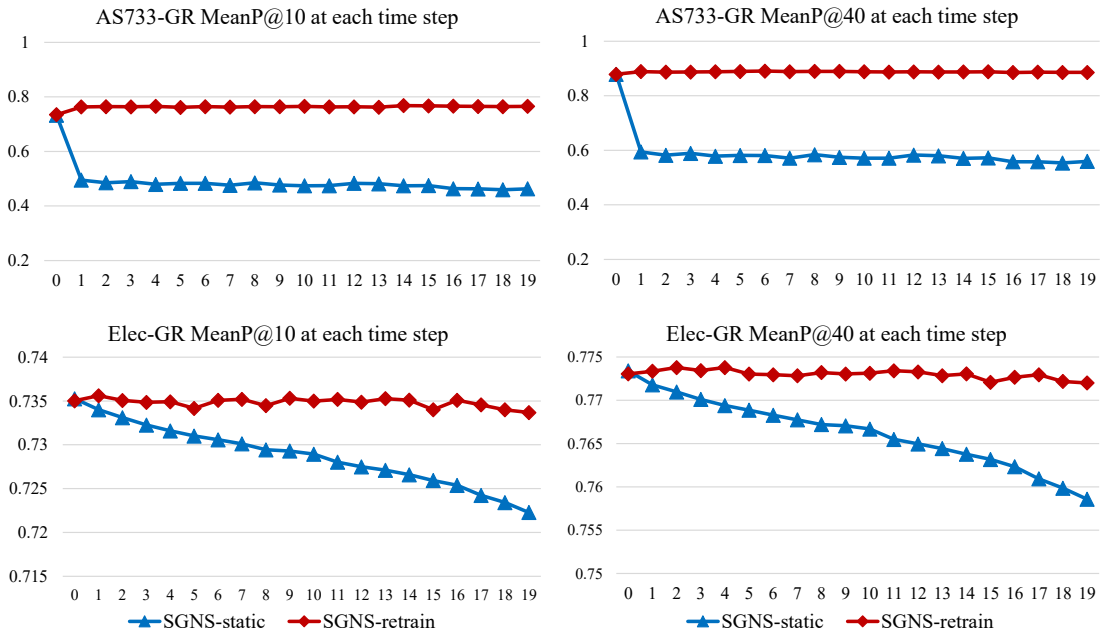


Figure 4.3: SGNS-static vs SGNS-retrain in graph reconstruct tasks for showing the necessity of dynamic network embedding: y-axis indicates Mean$P@k$ scores; x-axis indicates timesteps; each point depicts the average result over 20 runs at a timestep.

According to Figure 4.3, SGNS-retrain outperforms SGNS-static on both datasets. For AS733, SGNS-retrain maintains the performance at a superior level all the time, whereas the performance of SGNS-static suddenly decreases at $t = 1$ and then maintains a poor level afterward. For Elec, SGNS-retrain maintains the performance at a superior level all the time,

whereas the performance of SGNS-static gradually decreases. The difference of sudden drops on AS733 and gradual drops on Elec is due to the fact that the network topology between consecutive timesteps on AS733 varies more severely than on Elec (see Section 4.4.1.1), so that the obtained node embeddings at $t = 0$ is less useful afterward. Consequently, it is needed to promptly update node embeddings at each timestep (i.e., the necessity of DNE) as what SGNS-retrain, i.e., the naive DNE method does.

4.4.3.2 Incremental Learning vs Retraining

Instead of SGNS-retrain, recent DNE methods often adopt the incremental learning paradigm by continuously training the previous model on a new training set. Another baseline–SGNS-increment thus follows Algorithm 3 but replaces all operations in lines 7-14 with $\mathcal{V}^t_{\text{sel}} = \mathcal{V}^t$. The difference between SGNS-increment and SGNS-retrain is whether they reuse the

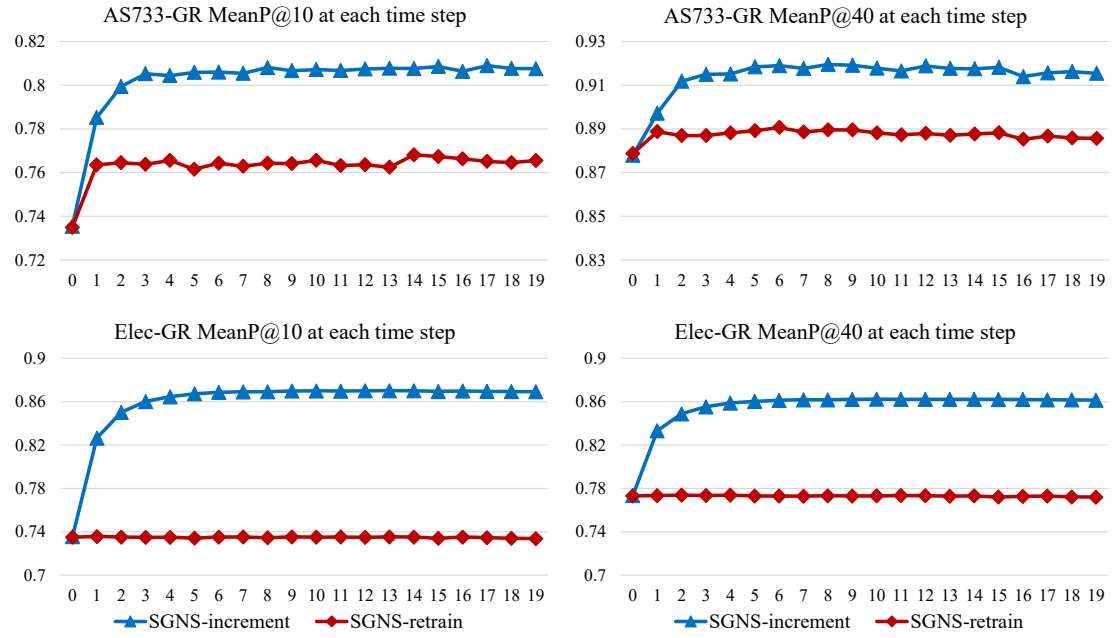


Figure 4.4: SGNS-increment vs SGNS-retrain in graph reconstruct tasks for showing the advantage of reusing previous models: y-axis indicates Mean$P$@k scores; x-axis indicates timesteps; each point depicts the average result over 20 runs at a timestep.

According to Figure 4.4, SGNS-increment outperforms SGNS-retrain on both datasets.

The general tendency on both datasets is the same, although the performances of SGNS-increment and SGNS-retrain are both less stable on AS733 than on Elec, due to the larger variations between consecutive snapshots on AS733 (see Section 4.4.1.1). These observations show that reusing the previous model as the initialization of next model might be not only useful for a dynamic network with small variations, but also useful for a dynamic network with relative large variations.

4.4.3.3 Visualization of Embeddings

To show how embeddings evolve over consecutive timesteps, we first employ a DNE method to obtain node embeddings with 128 dimensions, and then use Principle Component Analysis to project the node embeddings from 128 dimensions to 2 dimensions (so that they can be displayed on a 2 dimensional plane). As shown in Figure 4.5, GloDyNE keeps not only the relative position but also the absolute position of node embeddings between two consecutive timesteps, whereas SGNS-retrain cannot keep the absolute position (notice the rotation of the 'v' shape). The reason of why GloDyNE can well keep the absolute position is owing to the incremental learning paradigm which acts as an implicit smoothing mechanism.

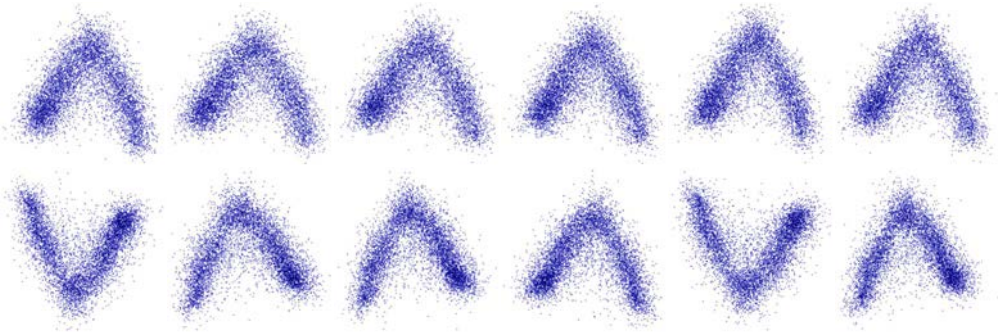


Figure 4.5: GloDyNE vs SGNS-retrain in embedding visualization tasks for showing the implicit smoothing mechanism of GloDyNE. The first row of six sub-figures is for applying GloDyNE on Elec to obtain node embeddings for six consecutive timesteps from 8 to 13 respectively. Similarly, the second row is for SGNS-retrain. To visualize embeddings, we further project them from 128 dimensions to 2 dimensions.

4.4.3.4 Different Node Selecting Strategies

According to Figure 4.3 and 4.4, the ranking among the three baselines is SGNS-increment > SGNS-retrain > SGNS-static. Although SGNS-increment (i.e., GloDyNE with $\alpha = 1.0$) achieves the best performance, it is not efficient enough since all nodes in a current snapshot are selected for conducting random walks and then training the SGNS model. One natural idea for further improving the efficiency is to select some representative nodes as the *approximate solution*, such that it can significantly reduce the wall-clock time but meanwhile, still retain a good performance. Consequently, in this work, we propose a node selecting strategy, denoted as $S4$, as described in Section 4.3.2.1 and Section 4.3.2.2.

To show the advantage of $S4$ used in GloDyNE, the baselines with different node selecting strategies are used for comparison. For fairness, the number of selected nodes at each timestep is set to $\alpha|\mathcal{V}^t| = 0.1|\mathcal{V}^t|$ for all strategies. Concretely, $S1$ selects the nodes randomly with replacement from reservoir $\mathcal{R}^t$ which records the most affected nodes; $S2$ selects the nodes randomly without replacement from $\mathcal{R}^t$ and then from all nodes in a current snapshot if $|\mathcal{R}^t| < 0.1|\mathcal{V}^t|$; $S3$ selects the nodes randomly without replacement from all nodes in a current snapshot. Intuitively, from the perspective of *diversity of selected nodes*, $S1 < S2 < S3 < S4$ because 1) sampling nodes from $\mathcal{R}^t$ cannot be aware of inactive sub-networks which exist in many real-world dynamic networks; 2) sampling nodes from all nodes in a current snapshot cannot guarantee the selected nodes have an enough distance from each other; 3) sampling one node from each sub-network after network partition as introduced in $S4$, however, can ensure the selected nodes have an enough distance from each other.

In order to compare the performance of GloDyNE with different node selecting strategies, the length of random walks l (see Section 4.3.2.3) should be also considered. Because as l increases, the generated random walks (or node sequences) become less distinguishable. An extreme case is that, if l goes to infinity, a random walker starting from any node in a network can well explore its global topology. As a result, we compare the four different node selecting strategies w.r.t. different ls as shown in Table 4.5.

According to Table 4.5, first, the overall ranking of the performance under a same l is

Table 4.5: The performance of GloDyNE with different node selecting strategies w.r.t. different length of random walks in graph reconstruction tasks. Each entry is obtained in the similar way as described in Table 4.1.

	AS733				Elec			
	$S1$	$S2$	$S3$	$S4$	$S1$	$S2$	$S3$	$S4$
l	Mean$P$@10							
3	15.275	18.841	20.623	**21.340**‡	04.038	05.912	06.105	**06.122**
5	36.791	38.657	39.671	**40.033**‡	07.091	11.204	11.519	**11.837**‡
8	43.976	44.289	44.807	**44.862**	10.518	14.886	15.357	**15.630**‡
10	44.714	44.934	**45.253**	45.225	12.029	17.446	18.059	**18.325**‡
15	48.083	48.311	48.495	**48.514**	17.704	25.468	25.963	**26.420**‡
20	53.713	54.083	54.323	**54.443**	26.656	34.437	34.790	**35.071**‡
30	63.585	63.881	64.163	**64.186**	44.556	48.710	48.938	**49.074**†
40	69.483	69.694	**69.961**	69.852	54.906	57.055	57.157	**57.220**
50	72.918	73.083	**73.324**	73.281	60.846	62.224	62.278	**62.306**
60	75.086	75.265	**75.507**	75.497	65.021	65.981	**66.031**	66.028
70	76.491	76.724	76.985	**77.046**	68.288	68.946	68.920	**69.004**†
80	77.723	77.982	**78.385**	78.208	70.825	71.272	71.332	**71.340**
90	78.778	79.090	79.367	**79.369**	72.916	73.257	73.268	**73.277**
100	79.846	79.991	80.227	**80.305**	74.621	74.882	74.870	**74.885**
l	Mean$P$@40							
3	22.097	26.784	28.791	**29.747**‡	04.972	06.587	06.820	**06.822**
5	52.509	54.063	54.695	**55.094**‡	05.925	11.482	11.936	**12.312**‡
8	59.334	59.606	59.809	**59.845**	07.532	14.409	15.183	**15.608**‡
10	60.522	60.871	61.005	**61.062**	09.278	18.567	19.401	**19.887**‡
15	66.123	66.801	**67.372**	67.340	20.117	32.013	32.617	**33.224**‡
20	72.998	73.506	74.077	**74.095**	35.458	44.739	45.027	**45.251**‡
30	81.225	81.617	82.111	**82.191**	57.477	60.500	60.604	**60.739**‡
40	85.083	85.506	86.007	**86.018**	66.260	67.436	67.488	**67.502**
50	87.332	87.705	**88.109**	88.105	70.602	71.218	71.209	**71.215**
60	88.756	89.102	89.327	**89.329**	73.414	73.731	**73.725**	73.714
70	89.713	89.913	90.085	**90.156**	75.493	75.584	75.564	**75.576**
80	90.474	90.648	**90.779**	90.695	77.046	77.014	77.016	**77.027**
90	91.103	91.217	91.259	**91.286**	78.300	78.214	78.179	**78.189**
100	91.667	91.608	91.703	**91.745**	79.287	79.150	**79.127**	79.126

$S1 < S2 < S3 < S4$, which exactly matches the ranking of the diversity of selected nodes as discussed above. Second, as l increases, the four strategies become less distinguishable, which verifies the above analysis of four strategies with respect to l. And third, comparing AS733 and Elec, it shows that the superiority of $S4$ over other three node selecting strategies, is more obvious on Elec than on AS733. It suggests that using $S4$ with GloDyNE on a lager dataset (e.g., Elec is larger than AS733 as shown in Table 4.4) might gain more benefits.

4.4.3.5 The Free Hyper-Parameter

The hyper-parameter α, which determines the number of selected nodes at each timestep, is designed for freely trade-off between effectiveness and efficiency. We vary α from 0.1 to 1.0 with step 0.1, together with other four smaller values. Each bar in Figure 4.6 has two results. The blue one shows the effectiveness which is measured by the mean of Mean$P@k$ over all timesteps and over 20 runs, while the red one shows the efficiency which is measured

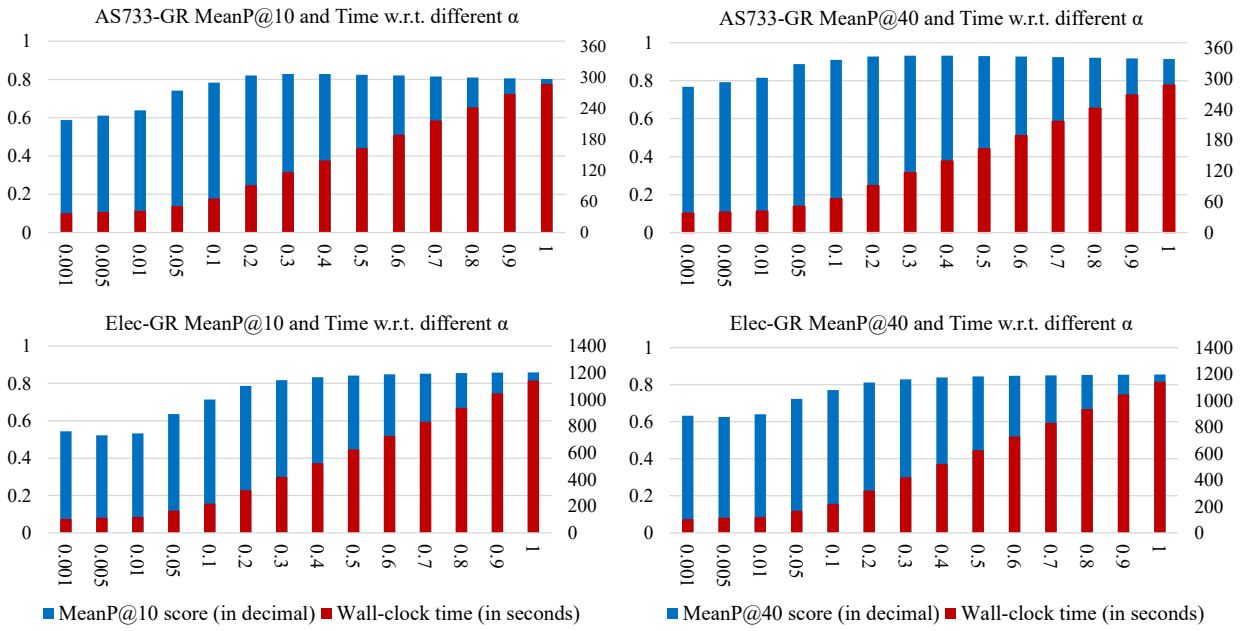


Figure 4.6: The effectiveness (in blue corresponding to the left y-axis) and efficiency (in red corresponding to the right y-axis) of GloDyNE w.r.t. different α (x-axis) which determines the number of selected nodes.

According to Figure 4.6, it demonstrates that the hyper-parameter α can be used to freely compromise between effectiveness and efficiency. With this free hyper-parameter, one could fulfill the real-world requirement by trade-off between effectiveness and efficiency, if downstream tasks require the latest node embeddings within a specified period. Besides, all experiments in the above sections set $\alpha = 0.1$, which implies one can obtain better results by increasing α at the risk of consuming more wall-clock time.

Furthermore, an interesting observation is that increasing α to a certain level achieves a very competitive performance as $\alpha = 1.0$ (GloDyNE with $\alpha = 1.0$ is equivalent to SGNS-increment), but consumes much less wall-clock time. This observation also supports that GloDyNE especially the proposed node selecting strategy that selects partial nodes, makes a good approximation to SGNS-increment that selects all nodes for further computation.

4.5 Chapter Summary

In this chapter, to answer the RQ2 of the thesis, we presented our work [59], which proposed a novel DNE method for better global topology preservation[10].

Specifically, the proposed DNE method called GloDyNE aimed to efficiently update node embeddings while better preserving the global topology of a dynamic network at each timestep, by extending the SGNS model to an incremental learning paradigm. In particular, unlike all previous DNE methods, a novel node selecting strategy was proposed to diversely select the representative nodes over a network, so as to additionally considers the inactive sub-networks for better global topology preservation. The extensive experiments not only confirmed the effectiveness and efficiency of GloDyNE w.r.t. other six state-of-the-art DNE methods, but also verified the usefulness of some special designs or considerations in the proposed method.

From a high-level view, GloDyNE can also be seen as a general DNE framework based on the incremental learning paradigm of SGNS model. With this framework, one may design

[10]See also the invited extended abstract [60] to be presented at the 38th IEEE International Conference on Data Engineering (ICDE), TKDE poster track, 2022.

a different node selecting strategy to preserve other desirable topological features into node embeddings for a specific application. On the other hand, the idea of selecting diverse nodes could be adapted to other existing DNE methods for better global topology preservation. Besides, one more future work, according to Figure 4.6, is to further investigate why selecting partial nodes can receive almost the same performance or even the superior performance compared to selecting all nodes.

Chapter 5

Robust Dynamic Network Embedding via Ensembles

In chapter 4, we proposed a Dynamic Network Embedding (DNE) method for better global topology preservation. This DNE method, like other existing DNE methods, mainly aimed at improving the effectiveness by considering some additional topologies or properties for an already preprocessed input dynamic network.

Unfortunately, to our best knowledge, existing DNE works ignore the *uncertainties* in generating an input dynamic network. In fact, most real-world datasets of dynamic networks are given in the form of edge streams (i.e., an edge list where each edge has its timestamp), rather than directly given as the input dynamic network that consists of a sequence of network snapshots captured at each timestep *after preprocessing*. Comparing to static networks, dynamic networks have a unique character called the degree of changes, which can be used as an index to quantify a kind of dynamic character of an input dynamic network about its rate of streaming edges between consecutive snapshots. The degree of changes could be very different for different dynamic networks. However, it remains unknown if existing DNE methods can robustly obtain good effectiveness to different degrees of changes, *in particular* for different dynamic networks generated from the same dataset by different slicing settings. To answer this open question, we test several state-of-the-art DNE methods, and find that they might not be robust enough to different degrees of changes. It thus emerges our RQ3

"*how can we embed a dynamic network robustly to the degree of changes?*".

In order to answer the RQ3 of the thesis, according to our work [61], this chapter presents a robust dynamic network embedding method w.r.t. the degree of changes which is defined as the average number of streaming edges between consecutive snapshots spanning a dynamic network. The source code to reproduce this work is available at `https://github.com/houchengbin/SG-EDNE`

The organization of this chapter is as follows. Section 5.1 discusses the background and motivation of this work. Section 5.2 reviews the literature of DNE methods and distinguishes the proposed method from them. Section 5.3 formulates the DNE problem, describes the DNE method in details, and analyzes the complexity of the DNE algorithm. Section 5.4.1 presents the experimental settings including benchmark datasets, compared methods, and evaluation protocols. The experimental results of comparative study, ablation study, parameter sensitivity, and scalability test are discussed in Section 5.4.2. Finally, Section 5.5 summarizes this chapter.

5.1 Background

Network embedding (a.k.a. network or graph representation learning) has been widely applied to various fields such as social networks, biological networks, telecommunication networks, knowledge graphs, and drug discovery [11, 48, 51, 53, 135]. Most previous network embedding methods are designed for static networks, while the real-world networks are often dynamic by nature [92–94], namely dynamic networks in which edges and/or nodes may be added and/or deleted at each snapshot of a dynamic network. Due to the dynamic nature of many real-world networks and the advantage of network embedding in various fields, Dynamic Network Embedding (DNE) has recently attracted considerable attention. DNE aims to efficiently learn node embeddings for each current network snapshot at each timestep by preserving network topology using current and historical knowledge, so that the latest embeddings can facilitate various downstream tasks.

A dynamic network in terms of a series of snapshots (graph streams) is often assumed

to have smooth changes [26–32, 39], which serves as the input to DNE. However, the assumption of smooth changes over snapshots would not hold for all real-world scenarios. It is natural to ask if existing DNE methods can perform well for an input dynamic network without smooth changes. To quantify the smoothness of changes, we suggest an index called Degree of Changes (DoCs)[1], which describes the average number of streaming edges between consecutive snapshots spanning a dynamic network. Note that, DoCs might be a way to quantify and rank the smoothness of changes for different dynamic networks. And the smaller DoCs corresponds to the smoother changes.

The DoCs of an input dynamic network depends on not only the nature of a network (e.g., a citation network versus an email network) but also how we preprocess it (e.g., 10 streaming edges per timestep versus 100 streaming edges per timestep). The existing general-purpose DNE methods are usually benchmarked on several network datasets, each of which is preprocessed by one special slicing setting that leads to a dynamic network with a special DoCs. However, the dynamic network with the special DoCs might not really fit the real-world requirement (e.g., it may require a smaller DoCs to update embeddings more frequently, or a larger DoCs to reduce the updating frequency). It remains unclear whether DNE methods can still perform well for the dynamic network with a different DoCs due to a different slicing setting on the same dataset. This motivates us to investigate the robustness of DNE to different input dynamic networks with different DoCs, especially when they come from the same dataset[2].

Although there have existed quite a few DNE works [26–35, 37–46, 59], they have not considered the effect of different DoCs of an input dynamic network to DNE methods. As a result, these methods might not be robust enough to different DoCs even if the corresponding input dynamic networks come from the same dataset, which is shown in our comparative study. However, the robustness of DNE to different DoCs is a desirable characteristic, as this would improve the reliability and usability of the DNE method while applying it to unknown

[1]The DoCs here is a global index to quantify the smoothness of changes of a dynamic network over all snapshots or timesteps.

[2]We can more fairly compare and rank the smoothness of changes of dynamic networks using DoCs if they are generated from the same dataset.

real world applications. To this end, we aim to propose an effective and more robust (w.r.t.

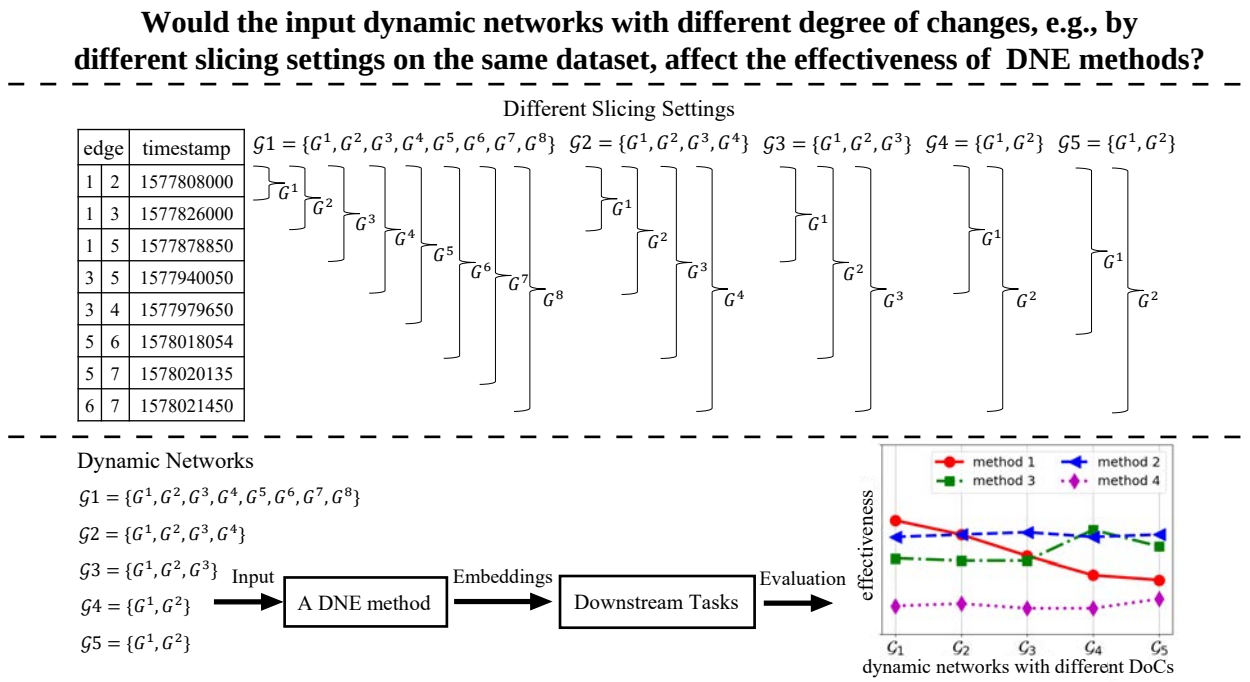


Figure 5.1: A toy example. The five input dynamic networks $\mathcal{G}1$,...,$\mathcal{G}5$ with different Degree of Changes (DoCs), constructed from the same dataset, are respectively fed to a DNE method and then evaluated by a downstream task. We could regard DoCs for $\mathcal{G}1$,...,$\mathcal{G}5$ as 1, 2, (3+2)/2=2.5, 4, 3 respectively. Despite the method 2 (in blue triangular) not always achieving the best, it may still be preferred due to its good effectiveness and robustness.

Concretely, the proposed method follows the notion of ensembles where the base learner adopts an incremental Skip-Gram neural embedding approach. Furthermore, a simple yet effective strategy is designed to enhance the diversity among base learners at each timestep by capturing different levels of local-global topology using random walks with different restart probabilities. Intuitively, the diversity enhanced ensembles encourage the base learners to learn from more diverse perspectives, such that it is more likely to have a part of (not necessarily all) base learners producing good embeddings at each timestep. The diversity enhanced ensembles therefore provide a better redundancy design to handle uncertainties (e.g., requiring different DoCs for different tasks) in generating a dynamic network and its time-evolving topological changes over snapshots of a dynamic network.

The contributions are as follows:

- An index called DoCs is suggested to quantify the smooth changes of a dynamic network. We then investigate the robustness of DNE to different input dynamic networks with different DoCs, especially when they are generated from the same dataset. The comparative study reveals that the existing DNE methods are not robust enough to DoCs, and might not always prefer an input dynamic network with the smaller DoCs, i.e., the smoother changes, which is often assumed in existing DNE works.
- An effective and more robust DNE method is proposed via the ensembles of incremental Skip-Gram embedding models at each timestep. To further boost the performance, we also propose a simple yet effective strategy to enhance the diversity among base learners by capture different levels of local-global topology.
- The comparative study also demonstrates the superior effectiveness and robustness of the proposed method compared to six state-of-the-art DNE methods. The ablation study and parameter sensitivity analysis confirm the benefits of special designs such as the ensemble design and the diversity enhancement strategy. The scalability test verifies our theoretical complexity analysis.

5.2 Prior Related Work

Most existing DNE methods are developed based on Static Network Embedding (SNE) methods. For SNE methods, there have been several good surveys [11, 48, 51, 53, 135]. In general, SNE methods can be divided into four main categories. First, the *Matrix Factorization (MF)* based approach [67, 68] encodes the desirable topological information into an n-by-n matrix, and then employs matrix factorization techniques to keep the top-d eigenvectors, which finally form the output embedding matrix. Second, the *Auto-Encoder (AE)* based approach [77, 78] also encodes the desirable topological information into an n-by-n matrix, and then feeds each row of input matrix to an auto-encoder for training. The middle layer of the trained auto-encoder gives output embeddings. Third, the *Skip-Gram (SG)* based approach [73, 74] encodes the desirable topological information via a sampling strategy (e.g., random walkers) into node sequences. The node sequences can be treated as the network language,

so that the Skip-Gram embedding model [69] can be adopted for learning node embeddings. And fourth, the *Graph Convolution (GC)* based approach [20, 136] defines node neighbors for each node by the first order proximity. For each node, node attributes are aggregated based on its neighbors so as to train a shared parametric mapping function. The trained function is finally used to infer node embeddings. Next, we review DNE based on the categories and methodologies of SNE.

MF based DNE. The key challenge is how to update previous factorization results based on the difference between current and previous input matrices. [29, 37] employ the matrix perturbation theory to resolve the challenge; [27] additionally considers the dynamic changes of node attributes; and [30] discovers the accumulated error of such incrementally updating mechanism and discusses when to restart the matrix factorization from scratch. Unlike these works, [26] and [33] iteratively minimize the reconstruction error of the current input matrix using previous factorization results (as a good initialization) for the faster convergence.

AE based DNE. One key challenge is how to fit the architecture of an Auto-Encoder for the current input matrix. When there is a new node leading to the increasing dimensionality of current input matrix, one neuron should be added to the first layer of the Auto-Encoder so as to make it runnable. [28] proposes a heuristic strategy to adjust the Auto-Encoder to handle the challenge. [37–39] also adopt an Auto-Encoder to incrementally learn node embeddings. But these works do not adjust the architecture of the Auto-Encoder and hence, they need to fix the node size in advance.

SG based DNE. The key challenge is how to generate appropriate training samples to continuously train the Skip-Gram model. [31, 34, 41, 42] determine most affected nodes based on the changes between current and previous snapshots, and then employ a sampling strategy to generate the training samples related to the most affected nodes. [59] additionally considers inactive subnetworks, i.e., no change occurs for several timesteps, and generates training samples for the inactive subnetworks to better preserve the global topology.

GC based DNE. Unlike above three categories, some SNE methods in this category, e.g., [20] can be directly applied to DNE problem as they aim to learn a shared parametric mapping function, which can be treated as a stacked deep neural network where the number

of neurons in the first layer is the dimensionality of node attributes. Therefore, there is no need to modify the neural network architecture even if there is a new node. It worth noticing that [43–46] additionally employ the attention mechanism or Recurrent Neural Network (RNN) to capture the temporal dependency.

Moreover, some DNE methods might not be solely classified into one of the above four categories. For instance, [35] applies a SNE method to obtain node embeddings at each timestep, and then feeds them sequentially to an RNN to capture the temporal dependency. [32] considers the triadic closure process, social homophily, and temporal smoothness in the objective function, and then optimizes it based on the existing edges of each snapshot.

Unlike the above DNE methods (for embedding a discrete-time dynamic network), [137] takes the input of edge streams (for embedding a continuous-time dynamic network), and defines a temporal awareness random walk sampling strategy to generate training samples for continuously training the Skip-Gram model. Other works about continuous-time dynamic network embedding can be found in the survey [94].

The proposed method, which takes snapshots as the input and Skip-Gram embedding approach as the base learner of ensembles, belongs to SG based DNE. But distinguished from existing works, this work aims to propose a more *robust* DNE method w.r.t. DoCs, and introduces the *ensembles* to DNE to improve its effectiveness and robustness.

5.3 Method

5.3.1 Problem Definition

Definition 1: A Static Network. $G(\mathcal{V}, \mathcal{E})$ denotes a static network where $\mathcal{V} = \{v_1, ..., v_n\}$ is a set of nodes; $\mathcal{V} \times \mathcal{V} \mapsto \mathcal{E}$ gives a set of edges; and $|\mathcal{V}|$ and $|\mathcal{E}|$ indicates the size of each set. The adjacency matrix is denoted as $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ where A_{ij} is the weight of edge e_{ij}; if $A_{ij} \neq 0$, the edge $e_{ij} \in \mathcal{E}$; and if $A_{ij} = 0$, there is no edge between v_i and v_j.

Definition 2: Static Network Embedding (SNE). Given a static network $G(\mathcal{V}, \mathcal{E})$, SNE aims to find a mapping $f : \mathcal{V} \mapsto \mathbf{Z}$ where $\mathbf{Z} \in \mathbb{R}^{|\mathcal{V}|\times d}$ is the output embedding matrix with a set

of node embeddings; each row vector $\mathbf{Z}_i \in \mathbb{R}^d$ corresponds to node v_i in $\mathcal{V}$; and $d \ll |\mathcal{V}|$ is user-specified embedding dimensionality. The objective is to best preserve network topology while learning node embeddings.

Definition 3: A Dynamic Network. $\mathcal{G} = \{G^0, G^1, ..., G^t, ...\}$ denotes the snapshots of a dynamic network at each timestep (i.e., graph streams). Each snapshot $G^t(\mathcal{V}^t, \mathcal{E}^t)$ is indeed a static network at timestep t. For a dynamic network, new nodes would come with new edges, and so this work only considers new edges but allows new nodes to occur.

Definition 4: Degree of Changes (DoCs). The DoCs is defined as the average number of streaming edges between consecutive snapshots spanning a dynamic network. Concretely, as the toy example shown in Figure 5.1, the DoCs for $\mathcal{G}1$,...,$\mathcal{G}5$ is (1*7)/7=1, (2*3)/3=2, (3+2)/2=2.5, 4/1=4, and 3/1=3 respectively. Although other definitions could exist, the definition here might be the straightforward one.

Definition 5: Dynamic Network Embedding (DNE). Given a dynamic network $\mathcal{G} = \{G^0, ..., G^t\}$ with the latest snapshot $G^t(\mathcal{V}^t, \mathcal{E}^t)$, DNE aims to find a mapping $f^t : \mathcal{V}^t \mapsto \mathbf{Z}^t$ where $\mathbf{Z}^t \in \mathbb{R}^{|\mathcal{V}^t| \times d}$ is the output embedding matrix with a set of node embeddings at timestep t; each row vector $\mathbf{Z}_i^t \in \mathbb{R}^d$ corresponds to node v_i in $\mathcal{V}^t$; and d is embedding dimensionality. The objective is to best preserve network topology and its dynamics into $\mathbf{Z}^t$, so that the latest node embeddings in $\mathbf{Z}^t$ can help various downstream tasks achieve good effectiveness.

Besides the effectiveness of DNE for a certain DoCs of an input dynamic network, this work also considers its robustness (i.e., a small standard deviation or variance of effectiveness) across different DoCs especially when the corresponding input dynamic networks are generated from the same dataset, as the toy example shown in Figure 5.1.

5.3.2 Method Description

The proposed method is termed as Skip-Gram based Ensembles Dynamic Network Embedding (SG-EDNE). It consists of four key components: Skip-Gram embedding approach, incremental learning paradigm, ensembles, and diversity enhancement as the overview illustrated in Figure 5.2.

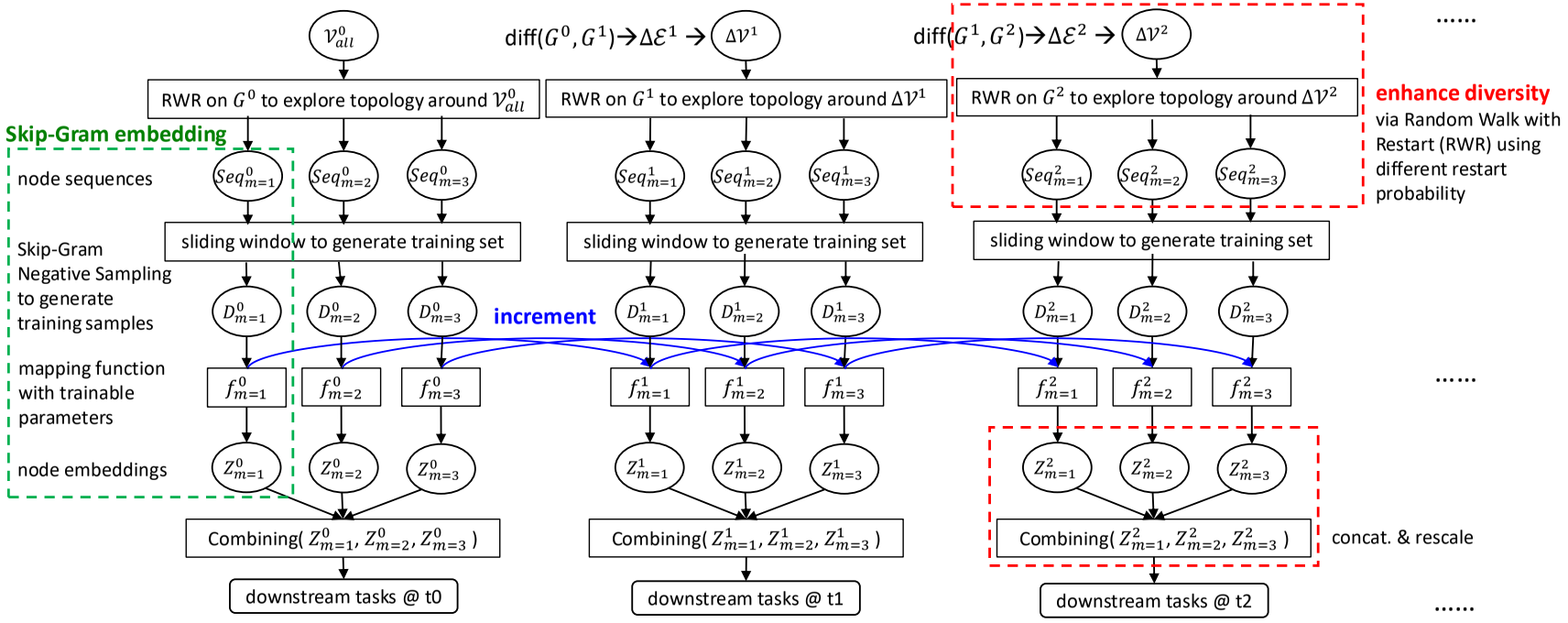


Figure 5.2: The overview of proposed method. At each timestep, multiple base learners (e.g., here are three) achieve ensembles. The base learner follows a Skip-Gram embedding approach. Between consecutive timesteps, each base learner inherits its previous base learner obeying an incremental learning paradigm. To enhance the diversity among base learners, random walk with restart using different restart probabilities is adopted to capture different levels of local-global topology (around the affected nodes by streaming edges). Embeddings from each base learner are concatenated and rescaled before downstream tasks.

5.3.2.1 Skip-Gram Embedding

A Skip-Gram embedding approach, an efficient neural network model [138], acts as the basic building block of the proposed method for learning node embeddings. At each current timestep t, the Skip-Gram Negative Sampling (SGNS) embedding model [59, 74] is adopted to learn node embeddings[3], given the node sequences that capture desirable network topology (how to generate node sequences is presented in Section 5.3.2.4).

Specifically, a sliding window with length $s + 1 + s$ is employed to slide along each node sequence, so as to generate a series of positive node-pair samples by $(v_{center}, v_{center+i})$, $i \in [-s, +s]$ and $i \neq 0$. In this way, the desirable network topology captured in node sequences is now encoded into node co-occurrence (v_c, v_i) statistics in D^t. We then maximize the log probability of node co-occurrence over all positive samples in D^t, i.e.,

$$\max \sum_{(v_c, v_i) \in D^t} \log P(v_i \mid v_c) \tag{5.1}$$

where v_c is the center node, and v_i is its nearby node with $1 \sim s$ orders proximity to v_c.

To define $P(v_i \mid v_c)$, it can be treated a binary classification problem [121]. Specifically, at timestep t, each positive sample $(v_c, v_i) \in D^t$ is distinguished from q negative samples $(v_c, v_{i'})$s where $v_{i'}$ is drawn from a unigram distribution P_{D^t} [121]. We then have $P(B = 1 \mid v_c, v_i) = \sigma(\mathbf{Z}_c^t \cdot \mathbf{Z'}_i^t)$ for observing a positive node-pair sample, and $P(B = 0 \mid v_c, v_{i'}) = 1 - \sigma(\mathbf{Z}_c^t \cdot \mathbf{Z'}_{i'}^t) = \sigma(-\mathbf{Z}_c^t \cdot \mathbf{Z'}_{i'}^t)$ for observing a negative node-pair sample. For all positive samples in D^t and their corresponding q negative samples, the objective is

$$\max_{\mathbf{Z}^t, \mathbf{Z'}^t} \sum_{(v_c, v_i) \in D^t} \{\log \sigma(\mathbf{Z}_c^t \cdot \mathbf{Z'}_i^t) + \sum_{i'=1}^{q} \mathbb{E}_{v_{i'} \sim P_{D^t}} [\log \sigma(-\mathbf{Z}_c^t \cdot \mathbf{Z'}_{i'}^t)]\} \tag{5.2}$$

[3]The reasons of using Skip-Gram rather than other embedding approaches are as follows. While extending SNE to DNE approaches, first, the matrix factorization based and auto-encoder based approaches often require to set a fixed number of nodes in advance [26, 29, 39, 40], and hence are hard to handle unlimited new nodes. Second, the performance of graph convolution based approach [20, 136] largely depends on the node attributes, which is out of the consideration of this work. Third, the Skip-Gram based approach has been shown to be not only effective but also efficient to DNE [31, 34, 41, 42, 59], and can easily handle node additions and deletions [59].

where $\sigma(x) = 1/(1+\exp(-x))$; operator $\cdot$ denotes a dot product between vectors; and $\mathbf{Z}_c^t \in \mathbb{R}^d$ is the embedding vector from row c of the trainable embedding matrix $\mathbf{Z}^t$ (between input and middle layers a.k.a. a project layer [69], and $\mathbf{Z}^t$ serves as DNE output after training), while $\mathbf{Z'}_i^t \in \mathbb{R}^d$ and $\mathbf{Z'}_{i'}^t \in \mathbb{R}^d$ are the embedding vector from column i or i' of another trainable matrix $\mathbf{Z'}^t$ (between middle and output layers) [120]. Eq. (5.2) is optimized by stochastic gradient ascent over D^t to learn these trainable embeddings. Intuitively, the more frequency two nodes co-occur in D^t, the closer or more similar their embeddings would be.

Note that, if the above approach only considers the snapshot at each timestep, i.e., treating as a static network embedding problem $\mathbf{Z}^t = f^t(G^t)$ at each timestep, we then need to retrain it from scratch at each timestep, which is time-consuming and cannot make use of historically learned knowledge.

5.3.2.2 Incremental Learning

The incremental learning paradigm is employed to avoid retraining from scratch for better efficiency and exploit historically learned knowledge for better effectiveness. To further improve efficiency, we consider one step back learned knowledge in input to middle layer $\mathbf{Z}^{t-1}$ and in middle to output layer $\mathbf{Z'}^{t-1}$ which are indeed the weights of SGNS neural network model, one step back snapshot G^{t-1}, and current snapshot G^t. Formally, the incremental Skip-Gram based DNE becomes

$$\mathbf{Z}^t = f^t(G^t, G^{t-1}, \mathbf{Z}^{t-1}, \mathbf{Z'}^{t-1}) \tag{5.3}$$

where trainable embeddings $\mathbf{Z}^t$ and $\mathbf{Z'}^t$ in the neural network model f^t are copied from its one step back $\mathbf{Z}^{t-1}$ and $\mathbf{Z'}^{t-1}$ in f^{t-1} (or denoted as $f^t \leftarrow f^{t-1}$) as the initialization before training. After training by Eq. (5.2), we can take out the updated embedding matrix $\mathbf{Z}^t$ as the output of DNE at timestep t.

It is worth noting that, Eq. (5.3) holds for $t > 0$ (online), and the latest two consecutive snapshots G^t and G^{t-1} provide the latest dynamics of streaming edges $\Delta\mathcal{E}^t$ (or given directly) which yields affected nodes $\Delta\mathcal{V}^t$ (see Figure 5.2). When $t = 0$ (offline), we have to train $\mathbf{Z}^0 = f^0(G^0)$ with randomized embeddings for all nodes in $\mathcal{V}^0$ of G^0 from scratch.

5.3.2.3 Ensembles

The ensembles have been shown useful to incremental learning problems on Euclidean data for handling uncertainties to improve model effectiveness and robustness [139–142]. Motivated by this, we attempt to employ ensembles to the DNE problem on graph (non-Euclidean) data.

At each timestep, we *separately* train several base learners of the incremental Skip-Gram based DNE model in Eq. (5.3) using the objective function in Eq. (5.2) respectively. The objective function of the ensembles at each timestep t becomes

$$\begin{aligned}\sum_{m=1}^{M}\{ \max_{\mathbf{Z}_m^t, \mathbf{Z'}_m^t} \sum_{(v_c,v_i)\in D_m^t} [\ \log\sigma(\mathbf{Z}_{m,c}^t \cdot \mathbf{Z'}_{m,i}^t) \\ + \sum_{i'=1}^{q} \mathbb{E}_{v_{i'}\sim P_{D_m^t}} [\log\sigma(-\mathbf{Z}_{m,c}^t \cdot \mathbf{Z'}_{m,i'}^t)]\ ]\ \}\end{aligned} \tag{5.4}$$

where the number of models to train is M; D_m^t is the training samples for base model m; and the output embedding matrix for base model m is $\mathbf{Z}_m^t = f_m^t(G^t, G^{t-1}, \mathbf{Z}_m^{t-1}, \mathbf{Z'}_m^{t-1})$. Please refer to Eq. (5.2) and Eq. (5.3) for other notations.

After training, the embeddings from each base learner are concatenated to form the unified embeddings, i.e.,

$$\mathbf{Z}^t = \mathbf{Z}_{m=1}^t \oplus ... \oplus \mathbf{Z}_{m=M}^t \tag{5.5}$$

where $\oplus$ denotes the concatenation operator[4]. Note that, we keep the dimensionality of unified embeddings to be d, i.e., $\mathbf{Z}^t \in \mathbb{R}^{|\mathcal{V}^t|\times d}$ regardless of the number of base learners M (but $d \geq M$). To achieve that, the dimensionality for each $\mathbf{Z}_m^t$ is $\mathbb{R}^{|\mathcal{V}^t|\times d_m}$ where d_m takes the integer of d/M and the reminder is added to d_m if $m = M$.

[4]The output from each base learner is embeddings in our (unsupervised learning) problem rather than a score in most supervised learning problems. Hence, the combining strategy of the outputs of embeddings is different from usual strategies, e.g., weighted voting of scores. Apart from concatenation, we also tested element-wise mean, max, min, and sum over each dimension of embeddings. The concatenation operator often obtained superior embeddings (evaluated by downstream tasks) for $d = 128$ by convention [59, 73, 74].

5.3.2.4 Diversity Enhancement

The diversity among base learners is recognized as a key factor to improve the performance of ensembles [139, 140, 143]. It is thus natural to enhance diversity for further boosting the effectiveness and robustness of the ensembles mentioned in Eq. (5.4). To be more specific, we propose to enhance the diversity among node sequences (and therefore training samples D_m^t) before feeding to each incremental Skip-Gram based DNE model. The strategy to enhance the diversity is that, each base learner adopts a different restart probability of Random Walk with Restart (RWR) to explore a different level of local-global topology over snapshot G^t starting from each node in $\Delta\mathcal{V}^t$ which is affected by streaming edges at timestep t.

For the restart probability R_m, a maximum $R^{\max}$ is set to assign different restart probabilities to M base learners by

$$R_1, R_2, ..., R_M \ = \ 0, R^{\max}/M, ..., (M-1)R^{\max}/M \tag{5.6}$$

where $R^{\max}$ and M are two hyper-parameters. The reason of setting restart probabilities uniformly is that, this work does not intend to find the optimal restart probabilities for a specific application. But one may try nonuniform restart probabilities for a certain real-world application. Furthermore, we believe different restart probabilities can capture different levels of local-global topology (see Figure 5.8). Considering two extremes: for $R_m = 1$, RWR can only explore very limited local neighbors around a starting node; for $R_m = 0$, RWR can collect more global information from a starting node.

With Eq. (5.6), a RWR using R_m for model m can be used to generate *node sequences* at timestep t. Concretely, for each node in $\Delta\mathcal{V}^t$ (or $\mathcal{V}^0$ if $t = 0$), r truncated RWRs with length l are conducted starting from it. For each RWR, the next node v_j much jump back to the starting node (i.e., restart) if a random number P_R from a uniform distribution $U[0, 1]$ meets $P_R < R_m$. Otherwise, the next node v_j is sampled based on the transition probability of its previous node v_i given by

$$P(v_j \mid v_i) = A_{ij}^t \ / \ \sum\nolimits_{v_{j'} \in \mathcal{V}^t} A_{ij'}^t \tag{5.7}$$

where A_{ij}^t is the edge weight between node v_i and node v_j at timestep t, and there is no edge between them if $A_{ij}^t = 0$.

Each base learner is trained separately to preserve a different level of local-global topology into embeddings. To keep the concatenated embeddings from M base learners in the same scale, we conduct a rescaling operation over each column[5] of $\mathbf{Z}^t$, which then serves as the input to downstream tasks. In this way, the diverse local-global topological information might be fairly preserved and used in downstream tasks.

5.3.3 Algorithm and Complexity

The workflow of proposed method SG-EDNE during online stage (i.e., $t > 0$) is summarized in Algorithm 4. Note that, Section 5.3.2.1 describes L6, L8; Section 5.3.2.2 describes L1, L7; Section 5.3.2.3 describes L2, L8, L9, L11; and Section 5.3.2.4 describes L3, L5, L12. For offline stage (i.e., $t = 0$), we still follow Algorithm 4 except that we have to train $\mathbf{Z}_m^0 = f_m^0(G^0)$ with random parameters for all nodes in $\mathcal{V}^0$ from scratch. One may also restart the algorithm after $t = 0$ to reduce potential accumulated errors [30]. For both online and offline stages, one can parallelize L4-L10, as there is no interaction among M base learners.

The ensembles of M base learners does not increase time complexity, as M is a small constant. Consequently, we ignore the effect of M as well as other constants such as r, l, and d in the following complexity analysis. During the *online* stage that follows the incremental learning paradigm as described in Algorithm 4, L5 requires $O(|\Delta\mathcal{V}^t|)$, as the algorithm conducts RWR for $|\Delta\mathcal{V}^t|$ nodes using alias sampling method [74]. The complexity of L6-L9 is $O(|\Delta\mathcal{V}^t|)$, since the algorithm generates training samples and accordingly trains SGNS based on $O(|\Delta\mathcal{V}^t|)$ node sequences. Moreover, the complexity for L1, L11, and L12 is $O(|\mathcal{V}^t|)$ respectively, while the complexity for L2 and L3 is $O(1)$ respectively. In summary, the overall complexity for the *online* stage is $O(|\Delta\mathcal{V}^t| + |\mathcal{V}^t|)$. While for the *offline* stage, i.e., $t = 0$ or in case of restarting algorithm, the overall complexity becomes $O(|\mathcal{V}^t|)$, since L5-L9 are now based on $O(|\mathcal{V}^t|)$ node sequences.

[5]Comparing to zero mean and unit variance scaling, we found [0,1] min-max scaling obtained better results in our cases, and is thus used in this work. The motivation of rescaling $\mathbf{Z}^t$ over each column is that we hope all d hidden variables/features from M base learners are fairly used in downstream tasks.

Algorithm 4 SG-EDNE at timestep t (online)

Input: two latest snapshots of a dynamic network G^{t-1}, G^t; previous trained models $f_{m=1}^{t-1}, ..., f_{m=M}^{t-1}$; number of base learners/models M; max restart probability $R^{\max}$; walks per node r; walk length l; sliding window size s; negative samples per positive sample q; embedding dimensionality d

Output: embedding matrix $\mathbf{Z}^t \in \mathbb{R}^{|\mathcal{V}^t| \times d}$

find affected nodes $\Delta\mathcal{V}^t$ by new edges based on G^{t-1}, G^t
assign dimensionality d_m to base learner based on d, M
assign restart probability R_m to base learner based on $R^{\max}$ and M by Eq. (5.6)
for base learner m=1 to M **do**
    generate node sequences Seq_m^t by RWR with r, l, R_m from each node in affected nodes $\Delta\mathcal{V}^t$
    generate training samples D_m^t by SGNS with s, q
    copy trainable parameters to f_m^t from f_m^{t-1}
    train f_m^t using D_m^t by Eq. (5.2)
    take out $\mathbf{Z}_m^t$ from f_m^t
combine $\mathbf{Z}_{m=1}^t$,...,$\mathbf{Z}_{m=M}^t$ to obtain $\mathbf{Z}^t$ by Eq. (5.5)
rescale $\mathbf{Z}^t$ over each column
return $f_{m=1}^t, ..., f_{m=M}^t$ and $\mathbf{Z}^t$

5.4 Experiments

5.4.1 Experimental Settings

5.4.1.1 Datasets

In this work, the independent variable to control is DoCs. Consequently, we adopt five different slicing settings to each dataset to generate five dynamic networks $\mathcal{G}1, ..., \mathcal{G}5$ with five different DoCs. The statistics are presented in Table 5.1. Given the same dataset, the more slices often lead to the smaller DoCs of the corresponding generated dynamic network.

All datasets[6] [144, 145] are originally in *edge streams* and each edge has an Unix timestamp. Because of a large volume of experiments and some DNE methods encountering the out of memory (OOM) issue, we take out partial edges up to date 19951231, 20070331, and 20050331 on Co-Author, FB-Wall, and Wiki-Talk respectively. For other datasets, their full datasets are used. For each dataset, five dynamic networks are generated in the following way: 1) order edge streams by timestamps; 2) take the first 1/5 edges to establish the initial snapshot G^0; 3) the rest of edges are roughly and evenly divided into 20, 40, 60, 80, and 100 slices respectively for dynamic networks $\mathcal{G}1$, $\mathcal{G}2$, $\mathcal{G}3$, $\mathcal{G}4$, and $\mathcal{G}5$; and 4) the largest connected component, unweighted, and undirected G^1 of $\mathcal{G}1$ is given by G^0 appending with the first slice of the 20 slices; the G^2 of $\mathcal{G}1$ is given by G^1 appending with the second slice of the 20 slices; and so on.

It is worth mentioning that there is no suitable well preprocessed dynamic networks to investigate the robustness of DNE to different DoCs. To achieve the goal, we utilize the

[6]DNC-Email (`http://networkrepository.com/email-dnc.php`) is the email network of Democratic National Committee email leak in 2016. College-Msg (`http://snap.stanford.edu/data/CollegeMsg.html`) is the messaging network of an online social network at the University of California, Irvine. Co-Author (`http://networkrepository.com/ca-cit-HepPh.php`) is the collaboration network of authors from the papers of high energy physics phenomenology in arXiv. FB-Wall (`http://networkrepository.com/ia-facebook-wall-wosn-dir.php`) is a social network for Facebook users interacting in their wall posts. Wiki-Talk (`http://snap.stanford.edu/data/wiki-Talk.html`) is the network of registered Wikipedia users editing in each other's talk pages for discussions.

unified preprocessing approach above to fairly deal with each real-world dataset, so as to generate several dynamic networks with different DoCs for each dataset.

Table 5.1: The statistics of datasets and the generated dynamic networks.

	DNC-Email	College-Msg	Co-Author	FB-Wall	Wiki-Talk
$\mathcal{G}$1-20slices	170.4	531.7	4269.9	1213.6	3951.4
$\mathcal{G}$2-40slices	85.2	265.8	2135.0	606.8	1975.7
$\mathcal{G}$3-60slices	56.8	177.2	1423.3	404.5	1317.1
$\mathcal{G}$4-80slices	42.6	132.9	1067.5	303.4	987.9
$\mathcal{G}$5-100slices	34.1	106.3	854.0	242.7	790.3
$\lvert\mathcal{E}\rvert^{init}$	959	3202	21298	7420	19602
$\lvert\mathcal{E}\rvert^{last}$	4366	13835	106696	31691	98630
$\lvert\mathcal{V}\rvert^{init}$	555	755	1842	4343	6782
$\lvert\mathcal{V}\rvert^{last}$	1833	1893	4466	10300	31263

[1] The upper block presents Degree of Changes (DoCs), as defined in Definition 4, for each generated dynamic network.

[2] The lower block shows the number of edges and nodes in the initial and last snapshots, which should be the same for the five dynamic networks generated from the same dataset.

Apart from real-world datasets, we also prepare *synthetic* datasets via the Barabási–Albert (BA) model [146] to verify the scalability of the proposed method. The BA model is used to generate the dynamic network such that the node degree (w.r.t. counts) distribution obeys the power law distribution (i.e., a scale-free network) at each timestep. Specifically, it implements the preferential attachment mechanism to successively add a new node with m_{BA} edges such that each edge connects the new node with higher probability to an existing node with larger node degree. The size of networks $|\mathcal{V}_{\text{BA}}|$ grows from a few nodes to *millions* of nodes. The details of scalability test are presented and discussed in Section 5.4.2.4

5.4.1.2 Compared Methods

The proposed method SG-EDNE (or EDNE for short) is compared to six state-of-the-art DNE methods. According to the literature review in Section 5.2, BCGD-G and BGCG-L [26] belong to the matrix factorization based DNE; DynLINE [31], GloDyNE [59] and SG-EDNE (this work) belong to the Skip-Gram based DNE; DynTriad [32] and tNEmbed [35] employ

other approaches to handle the DNE problem.

The source codes of BCGD[7], DynLINE[8], DynTriad[9], tNEmbed[10], and GloDyNE[11] are used in the experiments. In tNEmbed, the link prediction architecture is used to obtain node embeddings, since the datasets in this work do not have node labels. In GloDyNE, the parameter retains 0.1 by default to balance effectiveness and efficiency. BCGD-G and BCGD-L are two proposed algorithms in BCGD for the type 2 and 4 algorithms respectively. Regarding other parameters, we search {1e-4,1e-3,1e-2,1e-1} for 'l' in BCGD-G and BCGD-L; {3e-4,3e-3,3e-2,3e-1} for 'learn-rate' in DynLINE; {1e-3,1e-2,1e-1,1} and {1e-3,1e-2,1e-1,1} for 'beta-triad' and 'beta-smooth' in DynTraid; and {1,10,100,1000} for 'train-skip' in tNEmbed. We find the recommended parameters in the original source codes obtain better results in most cases (among the 30 cases: DNC-Email and College-Msg datasets, 5 dynamic networks per dataset, and 3 downstream tasks), and are thus adopted for all experiments.

Without otherwise specified, for all experiments, SG-EDNE employs following default parameters: number of base learners $M = 5$ and maximum restart probability $R^{\max} = 0.1$. For the parameters of Skip-Gram approach, we follow [59, 73, 74] so that walks per node r, walk length l, window size s, and negative samples q are set to 10, 80, 10, and 5 respectively.

For the fair comparison, the dimensionality of node embeddings for all methods is set to 128. Besides, all experiments are conducted given the following specification: Linux 4.15.0, 16 Intel-Xeon CPUs @2.20GHz, 256G memory, and Tesla P100 GPU with 16G memory.

5.4.1.3 Evaluation

The node embeddings at each timestep are taken out and stored in the same format for all methods, so that we can utilize the same evaluation protocol as shown in Figure 5.3.

Three types of downstream tasks are employed to evaluate the quality of node embeddings. Graph Reconstruction (GR) tasks [29, 39, 59] use current embeddings to reconstruct

[7] `https://github.com/linhongseba/Temporal-Network-Embedding`
[8] `https://github.com/lundu28/DynamicNetworkEmbedding`
[9] `https://github.com/luckiezhou/DynamicTriad`
[10] `https://github.com/urielsinger/tNodeEmbed`
[11] `https://github.com/houchengbin/GloDyNE`

snapshot (by retrieving node neighbors for every node) at *current* timestep. Node Recommendation (NR) tasks use current embeddings to recommend node neighbors for those affected nodes (by verifying their old and new neighbors) at *next* timestep. Link Prediction (LP) tasks [26, 32, 59] use current embeddings to predict new links (i.e., changed edges) at *next* timestep. For GR and NR tasks, we adopt cosine similarity between embeddings for ranking, and employ mean average precision at k (MAP@k) [28, 77] as the index. For LP tasks, we train an incremental logistic regression using previous new edges as positive samples and randomly sampled equal non-edges as negative samples, and employ area un-

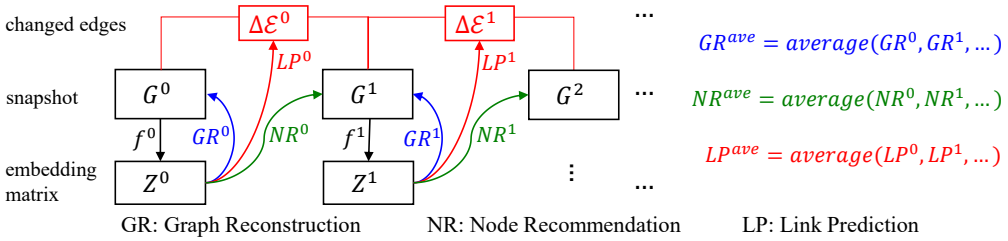


Figure 5.3: Evaluation protocol. For a task (e.g., GR), a method (e.g., SG-EDNE), and a dynamic network of a dataset (e.g., $\mathcal{G}1$ of DNC-Email), one result at each timestep t (e.g., GR^t) is measured by an index (e.g., $MAP@5$). For the dynamic networks $\mathcal{G}1$, $\mathcal{G}2$, $\mathcal{G}3$, $\mathcal{G}4$, and $\mathcal{G}5$ generated from the same dataset, there are about 20, 40, 60, 80, and 100 timesteps respectively. Each point shown in Figure 5.4, 5.5, and 5.6 depicts the averaged result over all timesteps (e.g., GR^{avg}), and we report the mean over 10 independent runs.

5.4.2 Results and Discussions

5.4.2.1 Comparative Study

The comparative study intends to compare the effectiveness and robustness of the DNE methods, which include the six state-of-the-art DNE methods and proposed method SG-

EDNE (or EDNE for short). In this work, the robustness is w.r.t. DoCs and is measured by the standard deviation (or stdev) over the five effectiveness results for the five input dynamic networks with different DoCs to a DNE method.

First, we observe the quantitative results from Table 5.2 that SG-EDNE achieves the best effectiveness and robustness (see mean±stdev) for most cases (25/30). The exceptions are GloDyNE in two NR tasks on Wiki-Talk, BCGD-L in two LP tasks on College-Msg, and BCGD-L in LP-AUC-L1-feature on Co-Author. Nevertheless, SG-EDNE is more robust w.r.t. DoCs (see stdev) in these exceptions, and still achieves the second best effectiveness (see mean). The same findings can be visually observed in Figure 5.4-5.6.

Second, from Figure 5.4-5.6 (effectivness below 40% are omitted), we observe that DoCs would greatly affect the effectiveness of some DNE methods such as DynTriad in GR tasks on College-Msg, tNEmbed in NR tasks on Co-Author, and BCGD-L in LP tasks on FB-Wall.

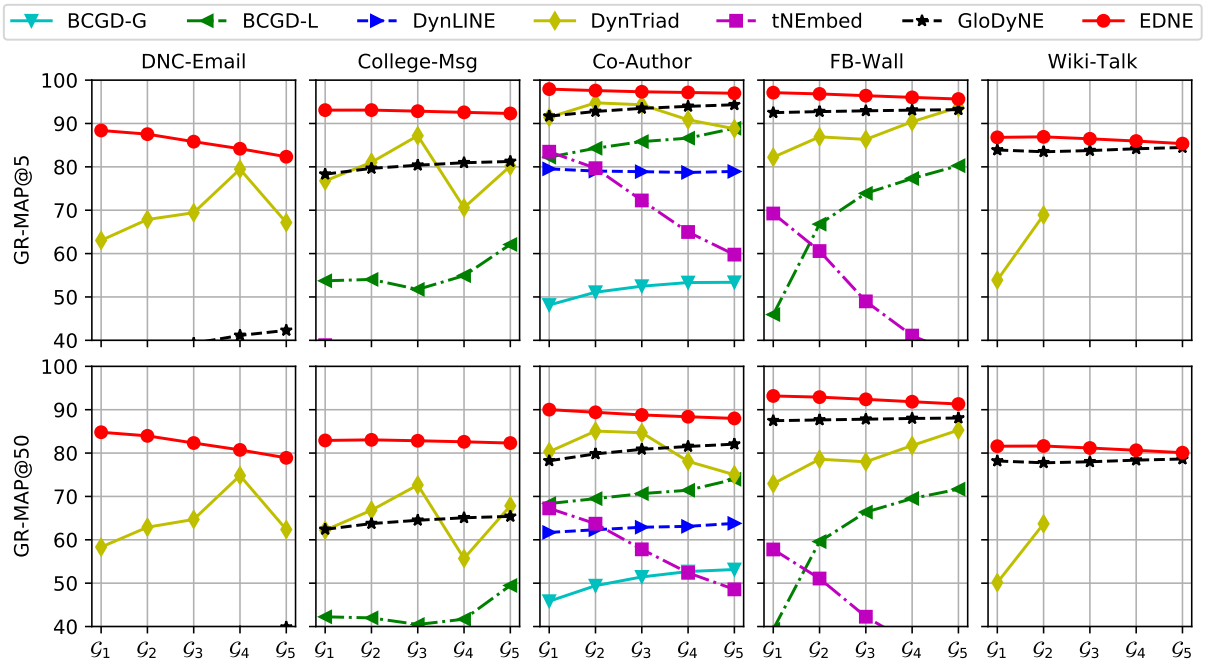


Figure 5.4: Comparative study for GR tasks. The evaluation protocol is illustrated in Figure 5.3. Each point depicts the mean over 10 independent runs of the averaged result over all timesteps, i.e., GR^{avg}. For each dataset, the ranking of DoCs of five input dynamic networks is $\mathcal{G}1 > \mathcal{G}2 > \mathcal{G}3 > \mathcal{G}4 > \mathcal{G}5$.

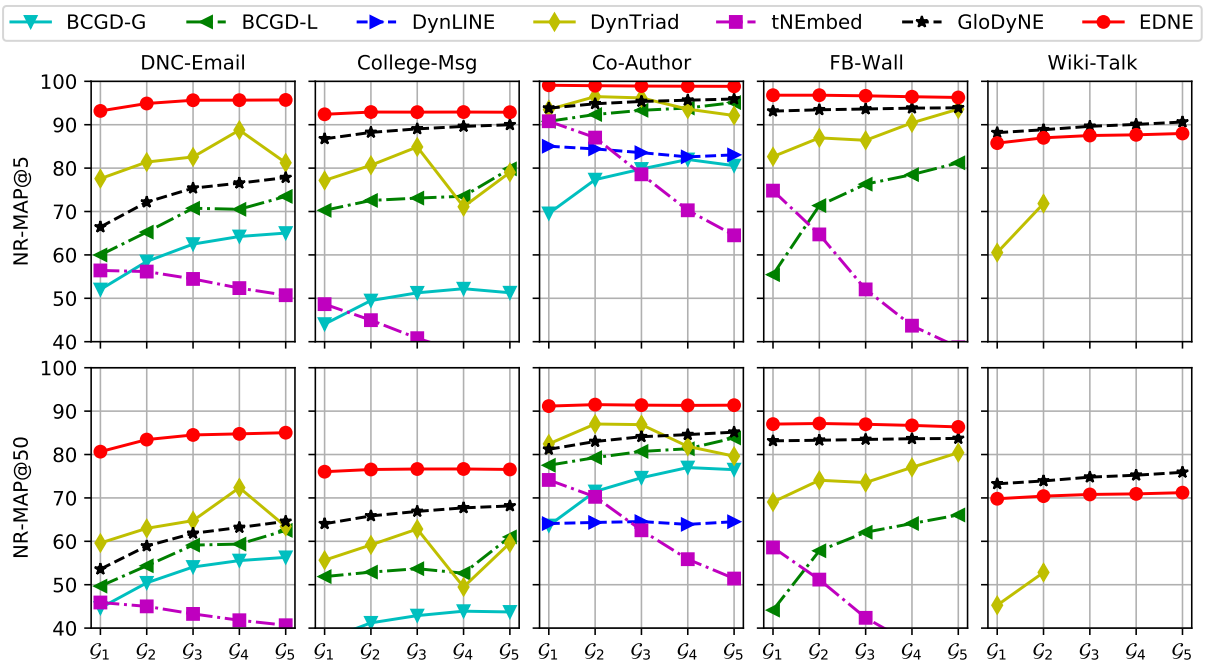


Figure 5.5: Comparative study for NR tasks.

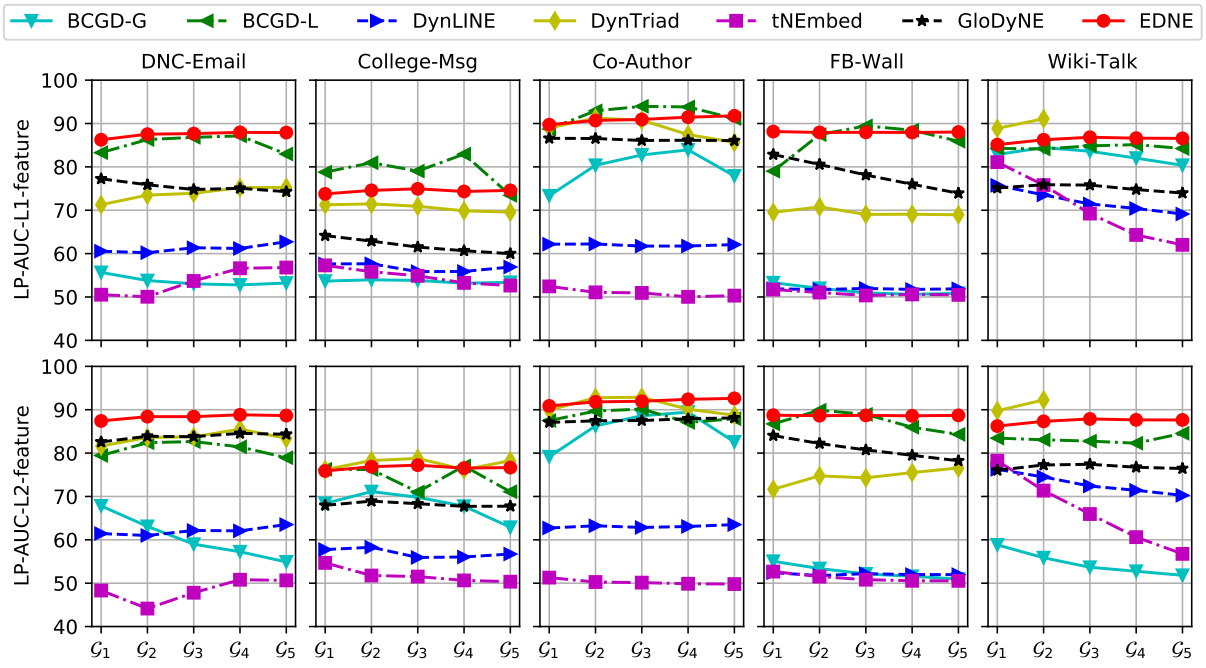


Figure 5.6: Comparative study for LP tasks.

Table 5.2: Quantitative results of comparative study for all tasks.

	DNC-Email	College-Msg	Co-Author	FB-Wall	Wiki-Talk
			GR-MAP@5		
BCGD-G	13.77±0.44	26.42±0.79	51.68±1.95	0.73±0.02	0.53±0.01
BCGD-L	22.57±0.78	55.3±3.55	85.57±2.24	68.84±12.3	3.37±1.09
DynLINE	2.64±0.11	2.79±0.07	79.02±0.28	5.42±0.10	0.22±0.02
DynTriad	**69.39±5.46**	79.15±5.44	92.03±2.24	87.92±3.90	OOM
tNEmbed	16.15±1.18	31.85±4.59	72.03±8.83	51.3±12.11	5.36±1.55
GloDyNE	38.19±3.78	**80.11±1.05**	**93.24±0.94**	**92.89±0.24**	**83.95±0.34**
SG-EDNE	†**85.65±2.20**	†**92.78±0.30**	†**97.39±0.34**	†**96.40±0.53**	†**86.30±0.58**
			GR-MAP@50		
BCGD-G	17.49±0.67	27.05±1.12	50.51±2.67	1.43±0.03	0.52±0.00
BCGD-L	21.29±0.78	43.18±3.20	70.78±1.90	61.31±11.7	2.90±0.95
DynLINE	2.42±0.06	2.97±0.06	62.76±0.71	5.45±0.06	0.25±0.02
DynTriad	**64.62±5.51**	**65.05±5.73**	**80.63±3.86**	79.31±4.11	OOM
tNEmbed	14.30±1.08	25.95±3.17	57.94±6.91	43.91±9.39	4.66±1.18
GloDyNE	36.15±3.41	64.23±1.07	80.49±1.36	**87.80±0.23**	**78.18±0.30**
SG-EDNE	†**82.14±2.14**	†**82.74±0.26**	†**88.92±0.72**	†**92.32±0.68**	†**81.02±0.59**
			NR-MAP@5		
BCGD-G	60.45±4.78	49.65±2.94	77.85±4.41	1.52±0.10	7.02±1.28
BCGD-L	67.99±4.81	73.85±3.19	93.08±1.45	72.57±9.15	24.73±5.90
DynLINE	14.80±0.74	5.05±0.29	83.73±0.89	7.99±0.14	1.37±0.09
DynTriad	**82.31±3.64**	78.56±4.52	94.37±1.68	88.01±3.72	OOM
tNEmbed	54.02±2.20	41.45±4.84	78.24±9.87	54.8±13.34	23.90±5.49
GloDyNE	73.67±4.08	**88.72±1.15**	**95.12±0.75**	**93.58±0.28**	†**89.47±0.87**
SG-EDNE	†**95.02±0.97**	†**92.80±0.20**	†**98.94±0.09**	†**96.60±0.20**	**87.18±0.78**
			NR-MAP@50		
BCGD-G	52.20±4.29	41.78±2.50	72.65±4.91	2.35±0.12	6.68±1.20
BCGD-L	57.01±4.50	54.42±3.34	80.53±2.10	58.85±7.85	18.67±4.18
DynLINE	11.75±0.30	5.32±0.17	64.28±0.25	7.95±0.05	1.67±0.11
DynTriad	**64.63±4.22**	57.35±4.55	83.57±2.92	74.83±3.79	OOM
tNEmbed	43.33±1.94	31.02±3.00	62.85±8.51	44.22±9.54	18.62±3.62
GloDyNE	60.42±3.90	**66.53±1.47**	**83.61±1.40**	**83.45±0.21**	†**74.62±0.94**
SG-EDNE	†**83.68±1.60**	†**76.50±0.23**	†**91.35±0.11**	†**86.84±0.28**	**70.63±0.49**
			LP-AUC-L1-feature		
BCGD-G	53.69±1.04	53.59±0.29	79.65±3.78	51.54±1.00	82.66±1.40
BCGD-L	**85.30±1.80**	†**79.03±3.17**	†**92.12±1.97**	**86.05±3.71**	**84.52±0.39**
DynLINE	61.21±0.86	56.78±0.80	61.99±0.21	51.84±0.10	72.05±2.34
DynTriad	73.83±1.46	70.61±0.76	88.83±2.08	69.47±0.67	OOM
tNEmbed	53.53±2.88	54.77±1.70	50.96±0.83	50.83±0.49	70.49±7.10
GloDyNE	75.45±1.04	61.84±1.51	86.29±0.23	78.29±3.17	75.11±0.71
SG-EDNE	†**87.47±0.63**	**74.44±0.39**	**90.92±0.70**	†**88.01±0.08**	†**86.28±0.61**
			LP-AUC-L2-feature		
BCGD-G	60.41±4.57	68.00±2.81	85.21±3.85	52.62±1.45	54.59±2.51
BCGD-L	80.97±1.53	74.32±2.68	88.47±1.22	**87.16±2.02**	**83.23±0.77**
DynLINE	62.04±0.85	56.95±0.93	63.08±0.28	52.07±0.22	72.97±2.15
DynTriad	83.58±1.24	†**77.53±1.11**	**90.88±1.66**	74.56±1.64	OOM
tNEmbed	48.34±2.42	51.79±1.54	50.28±0.54	51.22±0.81	66.58±7.63
GloDyNE	**83.83±0.69**	68.13±0.45	87.61±0.37	80.94±2.03	76.78±0.51
SG-EDNE	†**88.34±0.49**	**76.63±0.44**	†**91.93±0.61**	†**88.67±0.05**	†**87.34±0.58**

Each entry quantifies mean±stdev of each line (with five effectiveness scores) in Figure 5.4, 5.5, and 5.6. The top two are in bold, and † highlights the best one.

Third, it is interesting to observe there is no clear tendency that DNE methods prefer an input dynamic network with the smaller DoCs, i.e., the smoother changes, which is however assumed in several existing DNE works [26–32, 39].

Moreover, one more observation is that the performance of DNE methods might vary a lot for different datasets and different types of tasks. The reason is that different datasets and different types of tasks have different properties. For different datasets, the global (or average) clustering coefficient [147] of the last snapshot of DNC-Email is about 0.215, while that of Co-Author is about 0.665, and that of Wiki-Talk is about 0.114. For different types of tasks, GR using current embeddings to reconstruct current snapshot is a relative static task compared to LP for predicting future edges. Nevertheless, for the various datasets and tasks, the proposed method with default parameters can effectively and robustly outperforms other DNE methods for most cases.

5.4.2.2 Ablation Study

The ablation study aims to investigate if the key components in the proposed method (here termed as EDNE-rwr) are useful. EDNE-rw is a variant that replaces random walks with restart (rwr) used in EDNE-rwr with random walks (rw). EDNE-rw-fix is a variant that fixes one set of node sequences for all base models compared to EDNE-rw that generates different set of node sequences to each base model. DNE-rw is a variant that uses only one base model compared to EDNE-rw that employs more than one base models. EDNE-rwr-ws is a variant without the scaling operation compared to EDNE-rwr. The results for GR tasks are shown in Table 5.3, and each entry is obtained in the same way as that in Table 5.2. The results for NR and LP tasks can be found in Appendix A.4.

First, from the *diversity* perspective of the training samples to ensembles, EDNE-rwr > EDNE-rw > EDNE-rw-fix. The results among them indicate that enhancing the diversity among base models can improve the performance. Second, comparing the results of EDNE-rwr to EDNE-rwr-ws, it demonstrates the usefulness of the scaling operation.

Third, regarding DNE-rw, it is interesting to observe that applying ensembles to DNE by EDNE-rw-fix strategy would lead to the worse results, while other carefully designed

ensembles by EDNE-rw strategy and EDNE-rwr strategy often obtain the better results. One exception is that the proposed method by EDNE-rwr on Wiki-Talk is outperformed by DNE-rw without ensembles, though EDNE-rwr is more robust. This encourages us to further exploit a better design of ensembles for DNE methods as a future work.

Table 5.3: Ablation study of SG-EDNE for GR tasks.

	DNC-Email	College-Msg	Co-Author	FB-Wall	Wiki-Talk
GR-MAP@5					
DNE-rw	54.41±3.64	90.64±1.52	94.28±0.89	93.86±1.13	†**90.47±2.64**
EDNE-rw-fix	59.48±2.65	85.97±0.96	94.49±0.73	93.12±0.87	43.10±0.87
EDNE-rw	**83.35±1.22**	**92.57±0.21**	**97.06±0.33**	**95.47±0.44**	84.88±0.44
EDNE-rwr	†**85.65±2.20**	†**92.78±0.30**	†**97.39±0.34**	†**96.40±0.53**	**86.30±0.58**
EDNE-rwr-ws	82.29±3.32	92.17±0.78	96.96±0.49	94.89±0.75	85.49±0.93
GR-MAP@50					
DNE-rw	51.04±3.53	80.68±2.27	84.91±1.63	90.53±1.84	†**86.85±3.10**
EDNE-rw-fix	55.52±2.64	71.83±1.55	83.60±1.38	87.42±1.48	40.53±0.80
EDNE-rw	**79.75±1.22**	**82.36±0.20**	**88.34±0.66**	**91.44±0.55**	79.58±0.46
EDNE-rwr	†**82.14±2.14**	†**82.74±0.26**	†**88.92±0.72**	†**92.32±0.68**	**81.02±0.59**
EDNE-rwr-ws	77.72±3.32	78.23±0.87	86.62±1.08	88.89±0.95	80.32±0.90

5.4.2.3 Parameter Sensitivity

The parameter sensitivity analysis tries to investigate two important hyper-parameters M and $R^{\max}$. We vary M from 1 to 10 with step 1 and $R^{\max}$ from 0.1 to 0.9 with step 0.2. There are totally 10×5 points, and each point is given by the same evaluation protocol as illustrated in Figure 5.3. The results for GR tasks are shown in Figure 5.7, while the results for NR and LP tasks can be found in Appendix A.5.

First, for number of base models (or base learners) M, increasing M would not always improve performance, though $M > 1$ often achieves better performance. It seems different datasets have quite different preferences in M. Second, for the maximum restart probability $R^{\max}$, a smaller $R^{\max}$ is often preferred in most cases. Recall Eq. (5.6) for assigning different restart probabilities to RWR based on $R^{\max}$, which implies the proposed method employs a smaller restart probability than $R^{\max}$ in each base model. This motivates us to study the

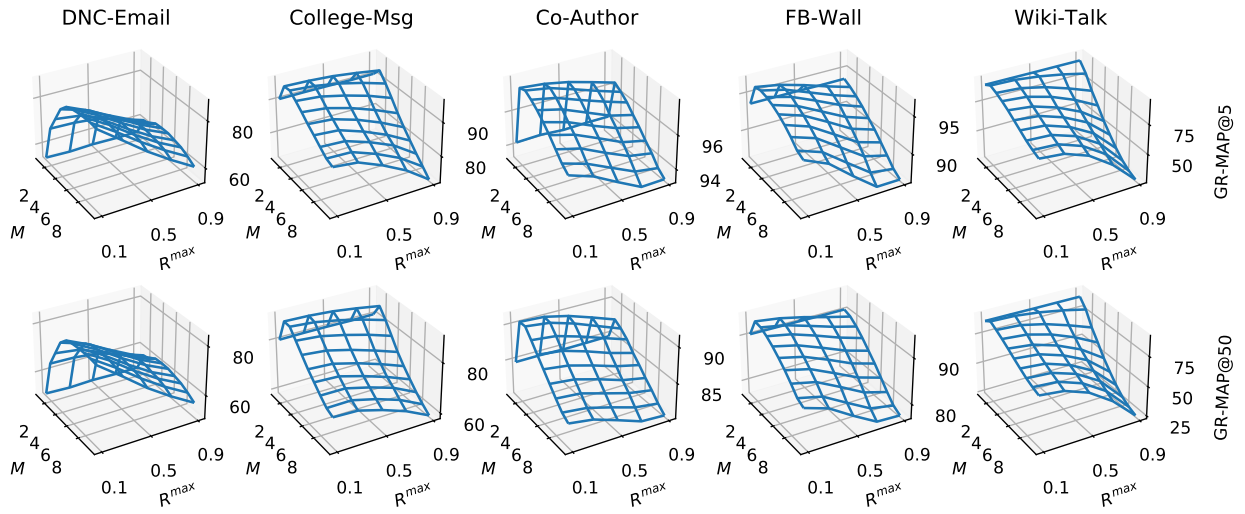


Figure 5.7: Parameter sensitivity for GR tasks. M is for the number of base models and $R^{\max}$ is for the maximum restart probability.

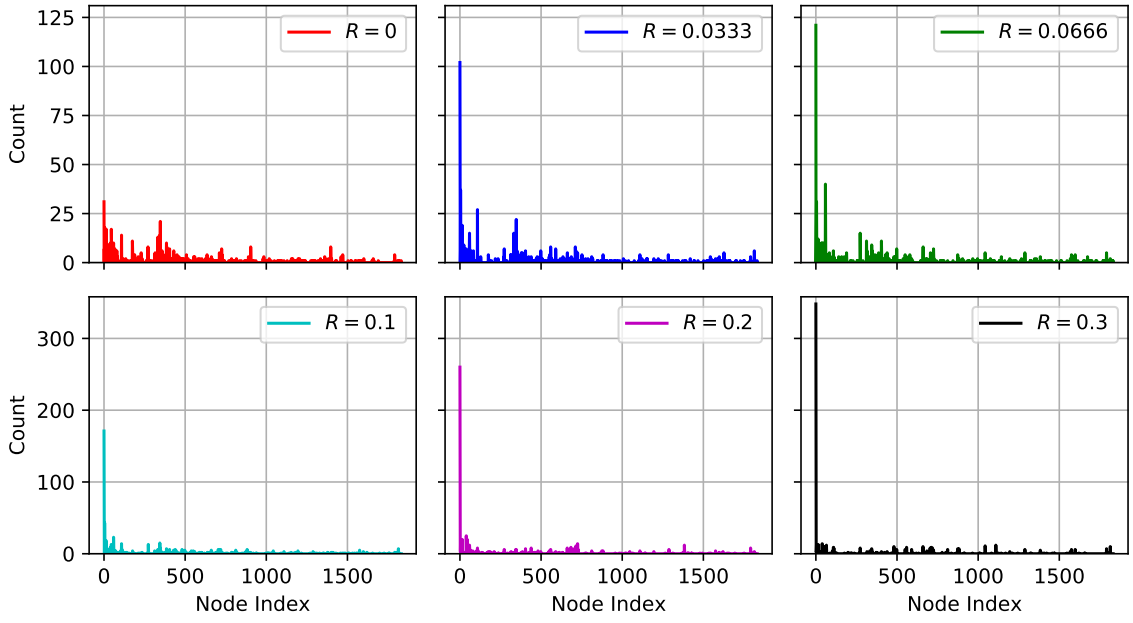


Figure 5.8: Statistics of RWR with different restart probabilities R starting from the same node on the last snapshot of DNC-Email.

property of RWR with different restart probabilities R, which is shown in Figure 5.8.

For Figure 5.8, we randomly pick a node and conduct RWR with different R starting from the same node on the last snapshot of DNC-Email. We observe that different Rs can explore different levels of local-global neighbors (i.e., different levels of local-global topology) of the starting node. In particular, a smaller R explores a richer set of neighbors (more evenly visit various nodes), while as R becomes larger, the *diversity* of the the neighboring information being explored is reduced (more likely revisit few nodes). This might be the reason why the proposed method prefers a smaller $R^{\max}$.

5.4.2.4 Scalability

This section presents the scalability test for SG-EDNE to verify its theoretical complexity given in Section 5.3.3. Since SG-EDNE consists of two stages namely the offline stage (at initial timestep or when retraining is needed) and the online stage (during incremental learning), two sets of datasets are generated via the BA model (see Section 4.1) with $m_{\text{BA}} = 4$ new edges per new node. For the offline stage, the number of nodes $|\mathcal{V}_{\text{BA}}|$ in each network ranges from 2^6 to 2^{20} with step 1 in the exponent. For the online stage, the number of newly emerging nodes $|\Delta\mathcal{V}_{\text{BA}}|$ per snapshot ranges from 2^2 to 2^{16} (lasting for 20 snapshots so the]

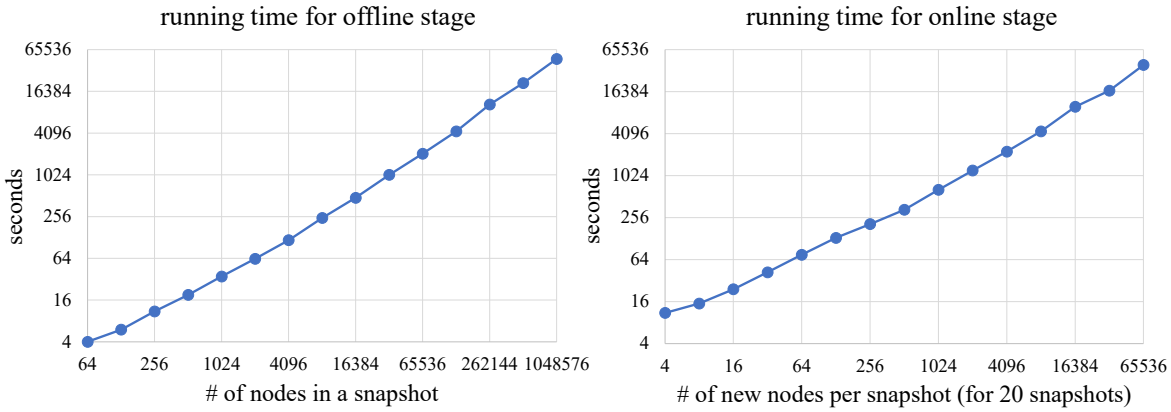


Figure 5.9: Scalability test for the proposed method. All axes are in $\log_2$ scale.

It might be easy to observe from Figure 5.9 that the results of scalability test are consis-

tent with the theoretical complexity. The running time for offline stage and online stage is linearly proportional to $|\mathcal{V}|$ and $(|\mathcal{V}^t| + |\Delta\mathcal{V}^t|)$ respectively.

5.5 Chapter Summary

In this chapter, to answer the RQ3 of the thesis, we presented our work [61], which proposed a robust DNE method w.r.t. the degree of changes.

Specifically, we suggested the robustness issue of DNE methods w.r.t. Degree of Changes (DoCs), and proposed an effective and more robust DNE method namely SG-EDNE basaed on the diversity enhanced ensembles of the incremental Skip-Gram model. The comparative study revealed that the existing DNE methods are not robust enough to DoCs, and also demonstrated the superior effectiveness and robustness of SG-EDNE compared to the existing DNE methods. The further empirical studies for SG-EDNE verified the benefits of special designs in SG-EDNE and its scalability.

We acknowledge that our first attempt to the robustness issue of DNE w.r.t. DoCs remains limitations. One can also preprocess a dynamic network with nonuniform number of streaming edges per timestep, and then study the robustness of DNE under this scenario. It might be another interesting but probably a more challenging scenario for the future work, and we refer readers to our work [63] for further discussions.. Moreover, it is also interesting to develop a better design of DNE ensembles, e.g., adopting other embedding model(s) as base learners, or exploiting a better strategy to enhance the diversity among base learners.

Chapter 6

Applications of Network Embedding

In Chapter 3-5, we have presented three novel Network Embedding (NE) methods [58, 59, 61], each of which tries to address the challenges caused by one type of more realistic networks. We employed some generic downstream tasks such as link prediction and node classification to evaluate the embeddings obtained by those NE methods.

In this chapter, we discuss NE from the perspective of applications. The applications of NE can be divided into two groups. One group is called generic applications (i.e., various downstream tasks used in Chapter 3-5 to evaluate embeddings), which are summarized in Section 6.2. Another group is called specific applications, which are discussed in Section 6.3. Particularly, Section 6.3 presents our work [62] about a specific NE application to drug-target interaction prediction, as a concrete example, to answer the RQ4 of the thesis "*how can we apply NE to drug-target interaction prediction?*".

6.1 Background

According to the literature review in Chapter 2 and prior related works in Chapter 3-5, a huge number of NE methods have been proposed to address various challenges resulting from various scenarios of networks. The reason why NE becomes such popular nowadays could be mainly owing to its widespread real-world applications. Regarding the development of

NE algorithms, some generic NE applications (or called downstream tasks in Chapter 3-5) such as link prediction and node classification [48–53] are employed to evaluate the quality of resulting embeddings. These generic NE applications can be treated as the *abstraction* of common (or standard) problems for mining or learning from networks, which paves a way for broader real-world problems.

Regarding a specific NE application to a specific real-world problem, one natural idea is to formulate the specific NE application as a well-studied generic NE application, and then solve it. For instance, one could formulate a drug-target interaction prediction problem as a link prediction problem, and apply NE to solve it [62]. There are a lot of specific NE applications to a wide variety of real-world problems, e.g., predicting unknown interactions in drug-target interaction networks [62], gene function prediction in biological networks [148], clustering vehicle trajectory in transportation systems [54], visualizing electronic health records of patients [149], recommendation system in social networks [150], traffic flow forecasting in road networks [151], fraud detection in financial networks [152], and so on. Note that, these specific real-world applications, which benefit business, scientific research, and our daily life, further motivate and boost the development of NE algorithms.

6.2 Generic Applications

One commonly used methodology to apply NE to a specific real-world problem is to first formulate the specific NE application as a well-studied generic NE application. Consequently, we first summarize the generic NE applications in this section, according to our works [58, 59, 61] presented in Chapter 3-5 as well as NE surveys [48–53].

Assume that node embeddings have been obtained by adopting a NE method to a given network. Based on the node embeddings, we then conduct the following downstream tasks or generic applications.

- Node Classification (NC). NC aims to assign the most likely labels to the nodes without labels [58, 59]. It can be seen as a multiclass classification problem. Concretely, we need to train a classifier, e.g., a one-vest-rest logistic regression, using the node embeddings

as input features and the known node labels as output labels. The trained classifier can be then used to assign the most likely labels to the nodes without labels.

- Link Prediction (LP). LP aims to predict missing links [58] or future links [59, 61]. It can be treated as a binary classification problem. Concretely, we should first obtain edge embeddings via a binary operator [61, 74] or simple concatenation [58, 59] between two node embeddings. If an edge exists, it would be labeled as a positive sample. And if an edge does not exist, it would be labeled as negative sample. Usually, an equal number of negative samples would be generated to balance the possible samples. After that, we need to train a binary classifier, e.g., a logistic regression, using the edge embeddings as input features and the edge labels as output labels. The trained binary classifier can be then used to predict missing links or future links.

- Node Recommendation (NR). NR aims to recommend the nearest neighbors of a given node. Unlike NC and LP above, NR does not require training. The search of k nearest neighbors can be directly conducted over the node embeddings. It is worth noting that NR would have different meanings under different contexts. For a static plain network or a snapshot of a dynamic plain network, NR tries to evaluate how well the topological information is preserved, which is also called graph reconstruction [59]. For a dynamic plain network, NR tries to evaluate how well the temporal topological information is preserved [61]. For a static attributed network, the goal of NR becomes to evaluate how well the topological information and attribute information are combined and preserved, e.g., as the specific application of paper recommendation [58] shown in Chapter 3.

- There are some other generic applications, e.g., visualization, community detection, anomaly detection, sub-network classification, and so on. We refer the readers to the NE surveys [48–53] for further reading. Overall, the generic NE applications can be at node level (i.e., the lowest level), edge level, sub-network level, or even whole network level. Regarding the input features to a generic NE application, the higher level embeddings could be calculated via some operators over their relative lower level embeddings, e.g., edge embeddings can be calculated via a binary operator between node embeddings as discussed in the link prediction above.

6.3 A Specific Application to Drug-Target Interaction Prediction

As mentioned in Section 6.1, there are a lot of specific NE applications to a wide variety of specific real-world problems, which benefits business, scientific research, and our daily life. In this section, we take the drug-target interaction problem, as an example, to demosrate how to apply NE to this specific problem and hence answer the RQ4 of the thesis. The main contents in Section 6.3 are adapted[1] from our work [62].

6.3.1 Background

Identifying novel Drug-Target Interactions (DTIs) is a crucial step in drug discovery [13]. Since experimentally determining DTIs is expensive and time-consuming [153], it is desirable to develop computational methods to identify promising candidate DTIs to accelerate the speed of drug discovery. Over the years, many computational methods have been proposed to identify novel DTIs. The existing methods could be divided into three categories.

The first category is the target structure based methods [154]. These methods simulate the docking process of drugs. However, the 3D structures of the proteins are required as the input, yet, the 3D structures of many proteins are unavailable. The second category is ligand similarity based methods [155]. These methods use the structural similarity between ligands to predict interactions, however, the sequence similarities between targets are ignored in predicting DTIs. The third category is machine learning based methods [56, 57, 156–162]. These methods often take the DTI network, drug structural similarity network (DSSN) and target sequence similarity network (TSSN) as the inputs to train a machine learning model for predicting DTIs. It should be noted that, the machine learning based methods have attracted a lot of research interests in recent years. Most of them focus on identifying novel

[1]Though I am the second author of our work [62], I participated deeply in this work including coming up with the idea of using implicit networks, providing advice of NE techniques, designing experiments, writing introduction part, and proofreading other parts. I have also gained the permission from the first author to reuse the materials of our work [62] in this thesis.

DTIs from known DTIs, as a drug might bind to more than one target [160] and vice versa, which could lead to successful drug repositioning.

Although many computational methods have been proposed to tackle the DTI prediction problem, the *implicit networks* extracted from the known DTI network are ignored in the existing works. However, it has been shown the implicit networks extracted from a bipartite network can improve the performance of the link prediction task in the bipartite network [163]. Besides, the implicit relations can also improve the performance of recommender systems [164]. Motivated by the success of using implicit networks or relations in other problems, we suggest to also consider the *implicit networks* extracted from the DTI network (a bipartite network) in the DTI prediction problem. The implicit networks, i.e., drug implicit network (DIN) and target implicit network (TIN) as shwon in Figure 6.1, are constructed using the second-order proximity in the DTI network. It is worth noticing that, an edge in DIN indicates that two drugs would bind to one common target, and an edge in TIN indicates that two targets would bind to one common drug, while DSSN (or TSSN) represents the structural similarity directly calculated between two drugs (or two targets). The topological structures of DIN and TIN would be different from the topological structures of TSSN and DSSN, as they are computed from different perspectives.

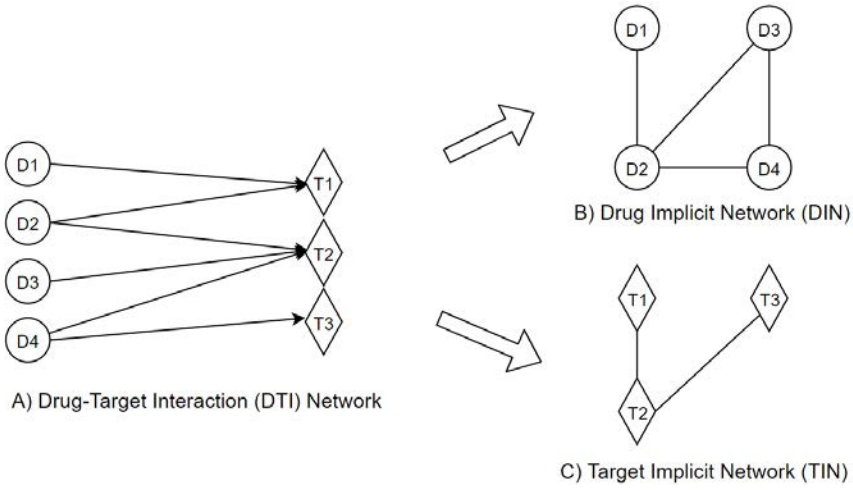


Figure 6.1: An example of the implicit networks construction from a DTI network. A circle represents a drug and a diamond represents a target. From the DTI network, the DIN and TIN are constructed based on the common neighbours.

To incorporate the implicit networks, we propose a method called Network Embedding based Drug-Target Interaction Prediction (NE-DTIP). Specifically, NE-DTIP is a machine learning based method and it includes two stages: the feature vector construction stage and the DTI classification stage. During feature vector construction stage, DIN and TIN are extracted from a DTI network. After that, drug embeddings and target embeddings are learned from DIN, TIN, DSSN, and TSSN using a network embedding method. *Unlike previous methods, the proposed method additionally considers two homogeneous networks, i.e., TIN and DIN, both of which are generated based on the implicit relations of a given DTI network.* During DTI classification stage, the drug-target pairs in a DTI network are regarded as positive samples, while randomly sampled unknown drug-target pairs in the DTI network are regarded as negative samples. The feature vector of each training sample is constructed via concatenating the corresponding drug embeddings and target embeddings. Finally, all samples are used to train a classifier for DTI predictions. The source code to reproduce this work is available at `https://github.com/BrisksHan/NE-DTIP`

We employ five real-world DTI datasets to evaluate the performance of NE-DTIP. Comparing to four state-of-the-art methods, NE-DTIP outperforms the existing methods on three out of five datasets. We also conduct a case study to verify the top twenty DTI predictions by NE-DTIP on the latest dataset, and it is interesting to find that six out of twenty novel DTIs (i.e., new DTIs not recorded on the dataset) are supported by recent studies.

6.3.2 Formulating the specific application based on generic applications

According to the generic applications as discussed in Section 6.2, the DTI prediction problem is more suitable to be formulated as a *link prediction* problem, rather than other frequently formulated generic problems such as node classification and node recommendation. The reason is that a DTI network is a bipartite network in which we only need to predict whether a drug (one type of node) has interaction (a link) with a target (another type of node). Therefore, there is no need to assign labels for nodes without labels (node classification) or

recommend a node to another node with the same type (node recommendation). Next, we formally define the DTI prediction problem as a link prediction problem.

Definition 1: A Homogeneous Network. Let $G = (\mathcal{V}, \mathcal{E})$ be a homogeneous network (or simply called a network) where $\mathcal{V}$ denotes a set of $|\mathcal{V}|$ nodes belonging to the same type, and $\mathcal{E}$ denotes a set of $|\mathcal{E}|$ edges belonging to the same type. The adjacency matrix of G is denoted as $\mathbf{W} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ where W_{ij} is the weight of edge e_{ij} between a pair of nodes (v_i, v_j). It should be noteworthy that, G^{DIN}, G^{TIN}, G^{TSSN}, and G^{TSSN} are homogeneous networks, as all nodes in each network belong to the same type, e.g., all nodes in G^{DIN} are drugs.

Definition 2: A DTI Network. A DTI network is a bipartite network in which nodes with two different types can be divided into two sets and there is no edge between nodes in the same set. Let $G^{\text{DTI}} = (\mathcal{V}^{\text{D}}, \mathcal{V}^{\text{T}}, \mathcal{E}^{\text{DTI}})$ be a DTI network, where $\mathcal{V}^{\text{D}}$ denotes a set of drugs; $\mathcal{V}^{\text{T}}$ denotes a set of targets; and $\mathcal{E}^{\text{DTI}} \subseteq \mathcal{V}^{\text{D}} \times \mathcal{V}^{\text{T}}$ denotes interactions. There is no drug-drug interaction or target-target interaction in the DTI network. For any drug-target pair $(v_i^{\text{D}}, v_j^{\text{T}})$ where $i \in \{1, ..., |\mathcal{V}^{\text{D}}|\}$ and $j \in \{1, ..., |\mathcal{V}^{\text{T}}|\}$; the weight W_{ij}^{DTI} describes the strength of interaction between them; and $W_{ij}^{\text{DTI}} = 0$ if there is no interaction. The matrix $\mathbf{W}^{\text{DTI}} \in \mathbb{R}^{|\mathcal{V}^{\text{D}}|\times|\mathcal{V}^{\text{T}}|}$ stores all weights about drug-target interactions.

Definition 3: DTI Prediction Problem. The aim of DTI prediction is to infer a DTI prediction matrix $\mathbf{M} \in \mathbb{R}^{|\mathcal{V}^{\text{D}}|\times|\mathcal{V}^{\text{T}}|}$ for all drug-target pairs, given the inputs of a DTI network (DIN and TIN are two implicit networks generated from the DTI network), a DSSN, and a TSSN. The output of each entry in $\mathbf{M}$ should reflect the possibility of the existence of the interaction between a drug-target pair.

Definition 4: Link Prediction Problem. To make the DTI prediction problem easier to solve, we formulate it as a link prediction problem. Concretely, to infer a DTI prediction matrix $\mathbf{M} \in \mathbb{R}^{|\mathcal{V}^{\text{D}}|\times|\mathcal{V}^{\text{T}}|}$, we only need to ask what is the probability of a link between every drug-target pair (i.e., a link prediction problem), which yields every entry of the DTI prediction matrix $\mathbf{M}$ (i.e., a DTI prediction problem). Note that, though link prediction is often treated as a binary classification problem, here we want to further know the probability or the confidence level of a link between a drug-target pair.

6.3.3 Constructing networks from the original relational data

Unlike generic NE applications, the input networks are often directly given. However, the input networks need to be constructed from the original relational data, when we try to apply NE to a specific real-world problem. In this section, we thus introduce how to construct the input networks as discussed in the above definitions.

6.3.3.1 Construction of Implicit Networks for DIN and TIN.

The implicit networks DIT or $G^{\mathrm{DIN}} = (\mathcal{V}^{\mathrm{D}}, \mathcal{E}^{\mathrm{DIN}})$ and TIN or $G^{\mathrm{TIN}} = (\mathcal{V}^{\mathrm{T}}, \mathcal{E}^{\mathrm{TIN}})$ are extracted from the DTI network $G^{\mathrm{DTI}} = (\mathcal{V}^{\mathrm{D}}, \mathcal{V}^{\mathrm{T}}, \mathcal{E}^{\mathrm{DTI}})$. Both implicit networks are homogeneous networks, where each node in G^{DIN} represents a drug; each node in G^{TIN} represents a target; and each edge in either G^{DIN} or G^{TIN} indicates the two corresponding nodes share at least one common neighbor in G^{DTI}. Following [165], the edge weight in G^{DIN} and G^{TIN} can be respectively defined as

$$W_{ij}^{\mathrm{DIN}} = \sum_{k \in \{1,...,|\mathcal{V}^{\mathrm{T}}|\}} W_{ik}^{\mathrm{DTI}} W_{jk}^{\mathrm{DTI}} \quad \forall\, i, j \in \{1, ..., |\mathcal{V}^{\mathrm{D}}|\} \text{ and } i \neq j \tag{6.1}$$

$$W_{ij}^{\mathrm{TIN}} = \sum_{k \in \{1,...,|\mathcal{V}^{\mathrm{D}}|\}} W_{ik}^{\mathrm{DTI}} W_{jk}^{\mathrm{DTI}} \quad \forall\, i, j \in \{1, ..., |\mathcal{V}^{\mathrm{T}}|\} \text{ and } i \neq j \tag{6.2}$$

where W_{ij}^{DIN} denotes the edge weight for a drug-drug node pair $(v_i^{\mathrm{D}}, v_j^{\mathrm{D}})$ in DIN and W_{ij}^{TIN} denotes the edge weight for a target-target node pair $(v_i^{\mathrm{T}}, v_j^{\mathrm{T}})$ in TIN.

6.3.3.2 Construction for DSSN and TSSN.

The edge weight in TSSN or G^{TSSN} is calculated via applying smith-waterman algorithm to pairs of protein sequences [166], while the edge weight in DSSN or G^{TSSN} is calculated via applying either Tanimoto coefficient [167] or [168] to pairs of drug molecular structures. The DSSN and TSSN are very likely to be fully connected after the pairwise similarity calculation. However, many edges are not informative, since their weights are too small. Furthermore, the speed of network embedding process would be improved if the edges with small weights are removed. Based on these considerations, we introduce an edge density parameter $\alpha \in [0, 1]$

to only keep a certain percentage of edges with large weights. In this way, the DSSN and TSSN would reveal strong community structures according to [169]. Specifically, for both DSSN and TSSN, we rank edges by weights and then keep the top ($\alpha \times 100$) percent, e.g., $\alpha = 0.1$ means that only the edges with weights in the top 10 percent are kept.

6.3.4 Incorporating network embedding to the proposed method

As discussed in Section 6.2, the DTI prediction problem can be treated as a link prediction problem. For a link prediction problem, we can train a classifier to predict whether an edge would occur between two nodes. To this end, we should have the feature vector with its known label for some edges to train the classifier. In this section, we first introduce why and how to apply Network Embedding (NE) to obtain node embeddings, and then present how to construct the labeled training samples. Finally, we describe what type of the classifier is used for training and how to use the trained classifier to predict DTIs.

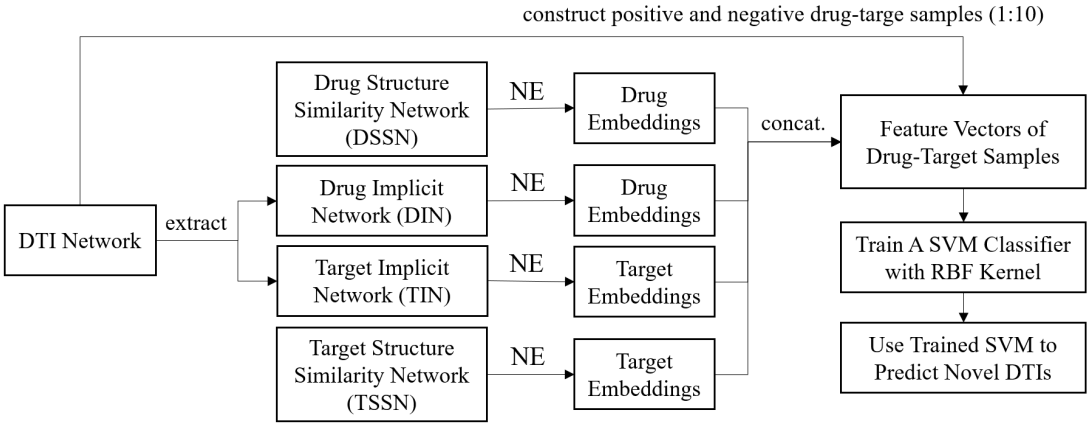


Figure 6.2: The framework of network embedding based DTI prediction method.

6.3.4.1 Network Embedding

Despite the weighted adjacency matrices of DIN, TIN, DSSN, and TSSN can be obtained according to Section 6.3.3. These matrices only reflect the first order proximity between nodes in each network, if we simply feed these matrices as the inputs to a classifier. It has been shown that the higher order proximities are also beneficial to the link prediction

problem [59, 74]. Therefore, a network embedding method is applied to encode both first order proximity and higher order proximities into low dimensional embeddings for nodes.

The network embedding method used here is a modified version of DeepWalk [58]. It has been introduced in Section 3.3.3.3, and thus is not repeated here. The reason of using it should be owing to its simplicity, efficiency, and effectiveness [51]. Note that, this network embedding method is applied to DIN, DSSN, TIN, and TSSN respectively to learn drug embeddings in $\mathbf{Z}^{\text{DIN}} \in \mathbb{R}^{|\mathcal{V}^{\text{D}}| \times d}$, drug embeddings in $\mathbf{Z}^{\text{TSSN}} \in \mathbb{R}^{|\mathcal{V}^{\text{D}}| \times d}$, target embeddings in $\mathbf{Z}^{\text{TIN}} \in \mathbb{R}^{|\mathcal{V}^{\text{T}}| \times d}$, and target embeddings in $\mathbf{Z}^{\text{TSSN}} \in \mathbb{R}^{|\mathcal{V}^{\text{T}}| \times d}$, where the embedding dimensionality d is kept the same for all types of embeddings.

6.3.4.2 Feature Vectors of Training Samples

The feature vectors of training samples are constructed based on the learned embeddings that are stored in $\mathbf{Z}^{\text{DIN}}$, $\mathbf{Z}^{\text{TSSN}}$, $\mathbf{Z}^{\text{TIN}}$, and $\mathbf{Z}^{\text{TSSN}}$. The positive sample is defined as the edge occurring in the DTI network. For the m-th drug-target pair $(v_i^{\text{D}}, v_j^{\text{T}}) \in \mathcal{E}^{\text{DTI}}$, which is labeled as $y_m^{\text{positive}} = 1$, the feature vector of a positive drug-target sample $\mathbf{Z}_m^{\text{positive}}$ is obtained by concatenating (denoted as $\oplus$) all corresponding embeddings, i.e.,

$$\mathbf{Z}_m^{\text{positive}} = \mathbf{Z}_i^{\text{TSSN}} \oplus \mathbf{Z}_i^{\text{DIN}} \oplus \mathbf{Z}_j^{\text{TSSN}} \oplus \mathbf{Z}_j^{\text{TIN}} \tag{6.3}$$

The negative sample is generated randomly from currently nonexistent edges in the DTI network where those edges, indicating drug-target interactions, have not be experimentally validated. For the n-th randomly sampled drug-target pair $(v_i^{\text{D}}, v_j^{\text{T}}) \notin \mathcal{E}^{\text{DTI}}$, which is labeled as $y_n^{\text{positive}} = -1$, the feature vector of the negative drug-target sample $\mathbf{Z}_n^{\text{negative}}$ can be obtained by in the same as described in Eq. (6.3).

The ratio of positive samples to negative samples is set to 1:10 according to [159], because there is less likely to have an interaction if we just randomly sample a drug-target pair.

6.3.4.3 DTI Prediction

As discussed in Section 6.3.2, to make the DTI prediction problem easier to solve, we formulate it as a link prediction problem. Note that, though link prediction is often treated as a

binary classification problem, here we want to further know the probability or the confidence level of a link between a drug-target pair.

In Section 6.3.4.2, the feature vectors of training samples, i.e., edge or drug-target pair embeddings with either positive label or negative label, were constructed. After that, we can train a classifier using these training samples. Specifically, the support vector machine (SVM) [170] is selected as the classifier, as it has been shown to be effective and robust [171]. To be more specific, the radial basis function (RBF) kernel of SVM [172] is chosen to offer a better non-linear classification ability without having to project features into a higher dimensional space. The SVM with RBF kernel is then trained using both positive and negative training samples.

With the *trained* SVM classifier, we can *infer* the DTI prediction matrix M, which tries to predict every entry of M or the probability of a link between every drug-target pair. The feature vector of a drug-target pair is constructed by concatenating node embeddings from DTN, DSSN, TIN, and TSSN as described in Section 6.3.4.2. Finally, the feature vector for every drug-target pair is fed, one by one, to the trained SVM with *Platt scaling* [173] to predict the probability of a link between every drug-target pair, until M is completed.

Algorithm 5 NE-DTIP: Network Embedding based Drug-Target Interaction Prediction

Input: DTI network; pre-computed DSSN; pre-computed TSSN; edge density parameter α; other hyper-parameters related to network embedding and SVM are stated in Section 6.3.5

Output: DTI prediction matrix $\mathbf{M} \in \mathbb{R}^{|\mathcal{V}^{\mathrm{D}}| \times |\mathcal{V}^{\mathrm{T}}|}$

1: construct DIN and TIN from DTI network ▷ Section 6.3.3.1
2: keep the top ($\alpha \times 100$) percent of edges in DSSN and TSSN ▷ Section 6.3.3.2
3: **for** G in $\{G^{\mathrm{DIN}}, G^{\mathrm{TIN}}, G^{\mathrm{TSSN}}, G^{\mathrm{TSSN}}\}$ **do**
4: obtain node embeddings for G by network embedding ▷ Section 6.3.4.1
5: construct feature vectors with labels as training samples ▷ Section 6.3.4.2
6: train a SVM classifier using training samples ▷ Section 6.3.4.3
7: obtain feature vectors for all drug-target pairs, and feed them one by one to the trained SVM with Platt scaling to infer every entry of $\mathbf{M}$, until $\mathbf{M}$ is completed ▷ Section 6.3.4.3
8: **return** DTI prediction matrix $\mathbf{M}$

6.3.5 Experiments and A Case Study

This section first states experimental settings such as datasets, compared methods with their hyper-parameters, and performance evaluation. After that, we compare the proposed method to other state-of-the-art methods, demosrate the effectiveness of using implicit networks, and analyze the sensitivity of the edge density parameter used in the proposed method. Finally, a case study is presented to show that the proposed method can predict the novel DTIs, which are not recorded on the original DTI dataset.

6.3.5.1 Experimental Settings

There are five DTI datasets for benchmarking. Four of them, i.e., Nuclear Receptor (NR), G-Protein-Coupled Receptors (GPCR), Ion Channel (IC), and Enzyme (E) come from [161], while the largest one, i.e., Drug-Target INhibition (DT-IN) comes from [56]. For all datasets, DSSN and TSSN are pre-computed. More specifically, the DSSN of NR, GPCR, IC, and, E is computed by following [168], while the DSSN of DT-IN is computed by Tanimoto coefficient [167]. The TSSN of all datasets are computed by Smith-Waterman algorithm [166]. Besides, DIN and TIN (implicit networks) for each dataset are extracted from each DTI network using the second order proximity. The statistics of five DTI datasets are shown in Table 6.1.

Table 6.1: The statistics of five DTI datasets.

	NR	GPCR	IC	E	DT-IN
# of DTI edges (in a DTI network)	90	635	1476	2926	4978
# of drug nodes (in both DTI and DIN networks)	54	223	210	445	732
# of targets nodes (in both DTI and TIN networks)	26	95	204	664	1915
# of implicit drug-drug edges (in a DIN network)	218	2748	2546	5137	25628
# of implicit target-target edges (in a TIN network)	54	668	8843	15497	46843

To demosrate the effectiveness of the proposed method NE-DTIP, it is compared to other four state-of-the-art methods, i.e., Netlaprls [174], BLM-NII [158], NRLMF [157] and NeoDTI [159]. There could exist other recent methods that require external inputs such as drug-drug interaction networks and target-target interaction networks, however, theses

external inputs are not available to some of above DTI datasets. As a result, we focus on the comparison among NE-DTIP, Netlaprls, BLM-NII, NRLMF, and NeoDTI. For fairness, all five compared methods can only take the inputs from the DTI network, DSSN, and TSSN.

Unless otherwise specified, the experiments obey the following hyper-parameter settings. The hyper-parameters of NE-DTIP consist of three parts. First, the hyper-parameters of embedding dimensionality, number of walks, walk length, window size, and negative samples for the incorporated network embedding method are set to 128, 10, 80, 10, and 5 respectively according to [59, 73, 74]. Second, we adopt the default hyper-parameters for the selected SVM with RBF kernel as recommended in [175]. Third, the density hyper-parameter α for DSSN and TSSN is set to 0.1 after hyper-parameter tuning. Apart from NE-DTIP, regarding the hyper-parameters of other four methods, we conduct a grid search around their default values to find the better hyper-parameters, which help these methods achieve the better performance. These hyper-parameters tuned on the largest dataset, i.e., DT-IN, are then used for other four smaller datasets.

For the experiments except the case study, we randomly and equally split all existing edges of a DTI network into 10 folds, in which each fold is treated as the testing set (10%) once, and the rest night folds are treated as the training set (90%) accordingly. Note that, this is known as the ten-fold cross validation. For both training and testing sets, the ratio of positive samples (from existing edges) to negative samples (from non-existing edges) is set to 1:10 according to [159], because there is less likely to exist an interaction if we just randomly sample a drug-target pair. To evaluate the performance, we follow [56, 159] to adopt the Area Under Precision and Recall curve (AUPR), as it is more suitable for the *imbalanced* classification problem where the ratio of positive samples to negative samples is 1:10. For the fairness evaluation, we report the average AUPR with its standard deviation over five independent ten-fold cross validations, i.e., 5×10 runs.

6.3.5.2 A Comparative Study

A comparative study of the proposed method NE-DTIP to other four state-of-the-art methods is conducted over five DTI datasets. The results are shown in Table 6.2.

Table 6.2: The AUPR scores on five benchmark datasets.

	Netlaprls		BLM-NII		NRLMF		NeoDTI		NE-DTIP	
	AUPR	stdev	AUPR	stdev	AUPR	stdev	AUPR	stdev	AUPR	stdev
NR	0.2288	0.0486	**0.3617**	0.0984	0.3336	0.0629	0.2308	0.0835	0.2906	0.0992
GPCR	0.4149	0.0358	0.4578	0.0434	**0.4979**	0.0392	0.4966	0.0624	0.4398	0.0628
IC	0.4704	0.0291	0.4763	0.0276	0.5201	0.0278	0.5841	0.0356	**0.6077**	0.0397
E	0.6930	0.0265	0.7299	0.0285	0.7352	0.0294	0.7844	0.0267	**0.7963**	0.0248
DT-IN	0.7816	0.0230	0.8024	0.0231	0.8484	0.0186	0.8560	0.0161	**0.8610**	0.0145

The best results are in bold. The stdev is the abbreviation for standard deviation.

From Table 6.2, we observe that NE-DTIP outperforms other methods over IC, E, and DT-IN datasets, despite it does not obtain the best performance over NR and GPCR datasets. Note that, it is impossible for a method to always outperform other methods over all datasets according to no free lunch theorem [176]. Considering the datasets themselves, we can observe from Table 6.1 that IC, E, and DT-IN have more implicit relations than NR and GPCR. Consequently, we believe the performance of NE-DTIP might be affected by the implicit edges in the input DTI network. And NE-DTIP would be more useful for a DTI network with rich implicit edges (i.e., the second order proximity relations).

6.3.5.3 Effect of Implicit Networks

To further investigate the effect of the implicit networks used in NE-DTIP, we compare the performance of NE-DTIP with implicit networks (i.e., the original method) and NE-DTIP without implicit networks (i.e., a variant of the original method) on DT-IN dataset. Specifically, the variant of NE-DTIP only takes DSSN and TSSN as the inputs, while the original NE-DTIP takes DSSN, TSSN, DIN, and TIN as the inputs. There are two variants of NE-DTIP called DSSN+TSSN (d=128) and SSN+TSSN (d=258). The later one tries to eliminate the effect of the dimensionality of edge embedding, which needs to concatenate node embeddings from each network and hence has 256×2 dimensions. It is equal to the dimensionality of edge embedding, which has 128×4 dimensions for the original NE-DTIP that takes two additional implicit networks.

Table 6.3: The effect of implicit networks. DIN and TIN are implicit networks.

DSSN+TSSN (d=128)		DSSN+TSSN (d=256)		NE-DTIP	
AUPR	stdev	AUPR	stdev	AUPR	stdev
0.8075	0.0171	0.8063	0.0172	0.8610	0.0145

NE-DTIP: DSSN+TSSN+DIN+TIN (d=128)

According to the results in Table 6.3, the original NE-DTIP with implicit networks outperforms its variants without implicit networks in both different dimensionality settings. It demosrates the effectiveness of using implicit networks in NE-DTIP.

6.3.5.4 Parameter Sensitivity

The proposed method NE-DTIP has an important hyper-parameter α called edge density parameter, which is used to control the edge density while constructing DSSN and TSSN. To analyze its sensitivity, α is set to 0.01, 0.04, 0.1, 0.4, and 1.0 with other parameters fixed.

Table 6.4: The effect of edge density parameter α in NE-DTIP.

α=0.01		α=0.04		α=0.1		α=0.4		α=1	
AUPR	stdev	AUPR	stdev	AUPR	stdev	AUPR	stdev	AUPR	stdev
0.8547	0.014	0.8608	0.0153	0.8610	0.0145	0.8536	0.0151	0.8491	0.0159

From the results in Table 6.4, we observe that NE-DTIP obtains the best two results when $\alpha = 0.04$ and $\alpha = 0.1$, while it obtains degraded results for other choices when α becomes either too small or too large. If α becomes too small, DSSN and TSSN become very sparse and thus there is no much topological structures to be encoded into node embeddings. If α becomes too large, DSSN and TSSN would keep many uninformative or noisy edges, which would also degrade the quality of node embeddings from DSSN and TSSN.

6.3.5.5 A Case Study

A case study on DT-In dataset is conducted to predict the novel DTIs that are not recorded on this dataset. Unlike previous experiments, all existing edges in DT-In network are used

for training, and there is no testing set. In fact, we exactly follow the procedures described in Section 6.3.4 and Algorithm 5, which finally infers a DTI prediction matrix M, in which each entry indicates the probability of a link between a drug-target pair. We then rank all these probabilities, and select the top-100 most likely links excluding known links (i.e., existing edges in DT-In network). The top-100 most likely links predicted by our method is plotted in Figure 6.3.

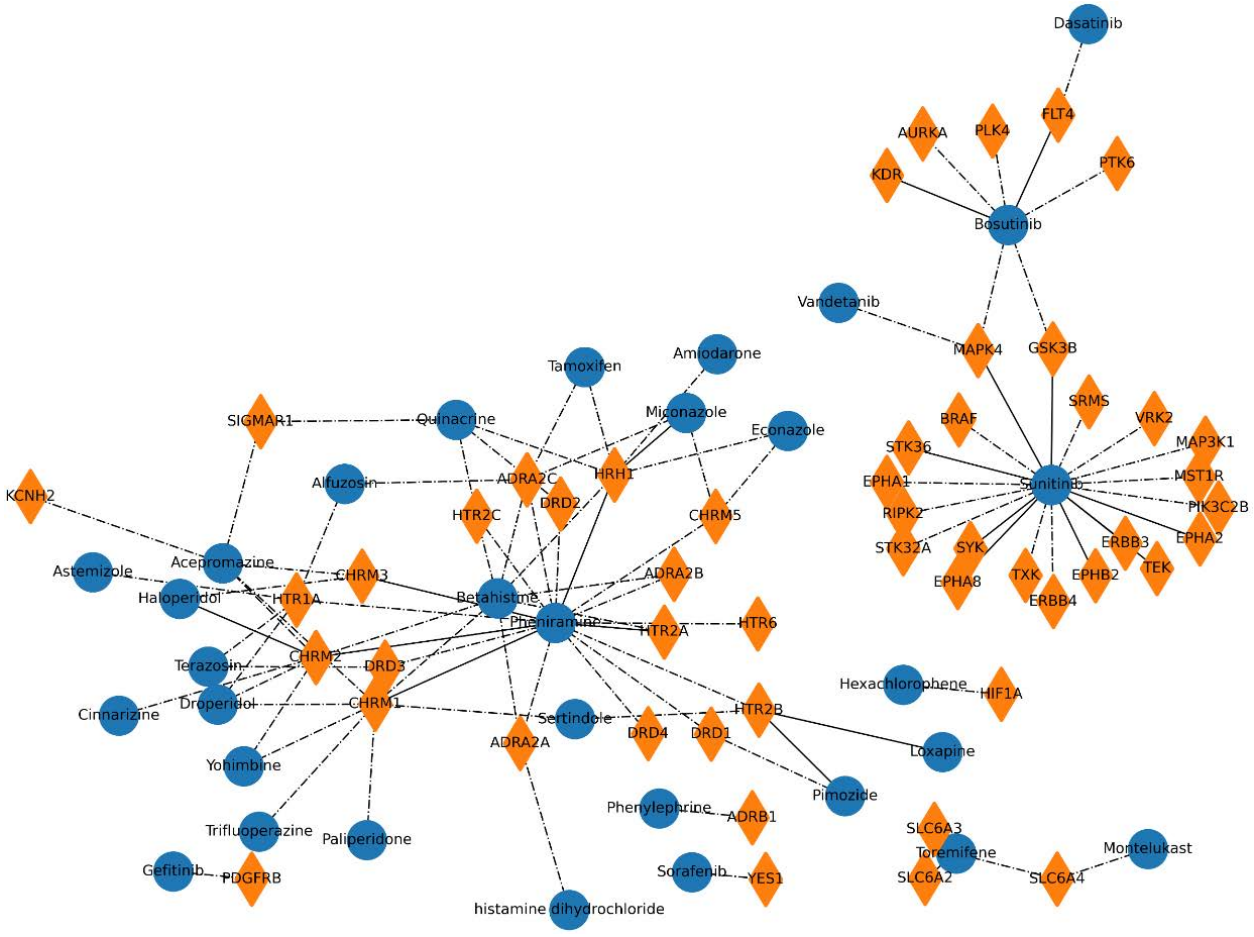


Figure 6.3: The top-100 novel DTIs predicted by the proposed method on DT-IN dataset. Each circle is a drug and each diamond is a target. The links between drugs and targets are the novel DTI predictions (i.e., new DTIs not recorded on the original dataset). The top-20 novel DTI predictions are drawn in solid lines, while the rest ones are drawn in dashed lines.

To interpret the novel DTI predictions in terms of drug discovery, we search for the supporting studies over the top-20 novel DTI predictions. It is interesting to find that six

of them are supported by recent studies. Sunitinib inhibits EPHB2 [177], SYK [178], and GSK3B [179]. Using Sunitinib as a treatment substantially increases ERBB3 [180]. Bosutinib is an inhibitor of KDR [181]. Haloperidol down regulates CHRM2 [182].

6.4 Chapter Summary

This chapter discussed NE from the perspective of applications, which can be divided into generic applications and specific applications. The generic applications, i.e., various downstream tasks used in Chapter 3-5 to evaluate embeddings, were summarized in Section 6.2. The specific applications were discussed in Section 6.3.

In particular, Section 6.3 presented our work [62] about a specific NE application to drug-target interaction prediction, as a concrete example, to answer the RQ4 of the thesis. Specifically, we proposed a NE based DTI prediction method. The experiments on five DTI datasets demonstrated that the proposed method is competitive with other four state-of-the-art methods, especially when there is a rich implicit relations in a DTI network. A further experiment about removing implicit networks (built using the implicit relations) suggested that there was a significant performance gain in incorporating the implicit networks, which however are ignored in previous works. More importantly, a case study on DT-In dataset indicated that the proposed method can predict the novel DTIs, which are not recorded on the original DT-In dataset but are supported by recent studies.

From this concrete example of a specific NE application, we could summarize a workflow of applying NE to a real-world problem.

- Formulate a given specific real-world problem into a well-studied generic NE application, e.g., a link prediction task or a node classification task.
- Construct a network or networks from the original relational data, and try to think about and utilize the characteristics of the network.
- Employ a suitable NE method to learn embeddings, which then serve as the inputs (or extra inputs) to a downstream task.

- Test and optimize the whole model, and try to conduct an ablation study to see whether the NE module really helps.

It is worth mentioning that, the specific NE applications to real-world problems, which benefit business, scientific research, and our daily life, further motivate and boost the development of NE algorithms. Regarding the future work, one may follow the above workflow to apply NE to various real-world problems, if one can reasonably construct a network or networks from the given problem and dataset.

Chapter 7

Conclusions

Networks (or graphs) are powerful and preferred data representation (or structure) to model the relationships between entities in many fields, e.g., social networks, knowledge graphs, molecular interaction networks from biology or chemistry, telecommunication networks, transportation networks, the Internet, recommender systems, sensor networks, and so on. Network Embedding (NE), serving as a kind of *feature extraction* technique for networks, can facilitate various downstream tasks and specific real-world problems. Although many successful NE methods are proposed, most of them are designed for embedding static plain networks. In fact, the real-world networks often come with one or more additional properties such as node attributes and dynamic changes. The central research question of this thesis was "*where and how can we apply NE for more realistic scenarios?*". It then resulted in four concrete research questions (RQs) as described in Chapter 1 and answered in Chapter 3-6 respectively, which is briefly summarized in Section 7.1. Furthermore, we provide several promising directions of the related future work in Section 7.2.

7.1 Thesis Summary

Chapters 1 and 2 mainly introduced the background of NE, suggested the RQs of this thesis, reviewed the literature of NE, and discussed other preliminaries. These build a foundation to systematically present our research works in the subsequent chapters.

In Chapter 3, we considered the (static) attributed network, which can better describe a real-world complex system where the relationships between entities can be represented as networks and the auxiliary information can be represented as node attributes. Attributed Network Embedding (ANE) aims to utilize network topology and node attributes to jointly learn enhanced low-dimensional node embeddings. However, prior ANE methods cannot effectively and efficiently embed attributed sparse networks which are important real-world scenarios. It leaded to our RQ1 "*how can we embed an attributed sparse network effectively, efficiently, and robustly?*". To answer RQ1, we first integrated network topology and node attributes to reconstruct an enriched denser network, and then learned node embeddings upon the denser network. In above two steps, the techniques such as Ball-tree K-Nearest Neighbors and random walks based Skip-Gram model were adopted to guarantee the scalability, which was demonstrated via theoretical complexity analysis. The extensive empirical studies showed the effectiveness and efficiency of the proposed method, as well as its robustness to different networks or the same network with different sparsities.

In Chapter 4, we moved to the dynamic (plain) network, which is a more realistic scenario as real-world complex systems often evolve over time. Dynamic Network Embedding (DNE) aims to efficiently learn node embeddings at each timestep by preserving network topology and its dynamics. Most prior DNE methods try to capture the topological changes at or around the most affected nodes and accordingly update node embeddings. Unfortunately, this kind of approximation, although can improve efficiency, cannot effectively preserve the global topology of a dynamic network at each timestep, due to not considering the inactive sub-networks that receive accumulated topological changes propagated via the high-order proximity. It produced our RQ2 "*how can we embed a dynamic network with global topology preservation?*". To answer RQ2, we proposed a novel node selecting strategy to diversely select the representative nodes over a network, which was coordinated with a new incremental learning paradigm of Skip-Gram based embedding approach. The experiments showed the proposed method achieved the superior or comparable performance w.r.t. the state-of-the-art DNE methods. Particularly, it significantly outperformed other methods in the graph reconstruction task, which demonstrated its ability of global topology preservation.

In Chapter 5, we made a step further towards robust DNE. Comparing to static networks, dynamic networks have a unique character called the degree of changes, which can be used as an index to quantify a kind of dynamic character of an input dynamic network about its rate of streaming edges between consecutive snapshots. The degree of changes could be very different for different dynamic networks. However, it remains unknown if existing DNE methods can robustly obtain good effectiveness to different degrees of changes, in particular for corresponding dynamic networks generated from the same dataset by different slicing settings. To answer this open question, we conducted a comparative study over six state-of-the-art DNE methods, and found they were not robust enough to different degrees of changes. It thus emerged our RQ3 "*how can we embed a dynamic network robustly to the degree of changes?*". To answer RQ3, the proposed method followed the notion of ensembles where the base learner adopted an incremental Skip-Gram embedding approach. To further boost the performance, a simple yet effective strategy was proposed to enhance the diversity among base learners at each timestep by capturing different levels of local-global topology. Extensive experiments demonstrated the benefits of special designs in the proposed method, and its superior performance compared to state-of-the-art DNE methods.

In Chapter 6, we applied NE to a specific real-world problem, rather than those generic downstream tasks as used in previous chapters for benchmark. The problem we chose was the Drug-Target Interaction (DTI) prediction problem, which is a crucial step in drug discovery and benefits human health. It yielded our RQ4 "*how can we apply NE to DTI prediction?*". To answer RQ4, the DTI prediction problem was formulated as a link prediction task. After that, we stated the way to prepare input networks from the original DTI dataset. Finally, we proposed a new NE based DTI prediction method, in which a NE method was applied to obtain node embeddings over each prepared network, and then we followed the paradigm of link prediction to build training samples, train a classifier, and make predictions. Unlike previous methods, the proposed method additionally considered two implicit networks generated from the DTI network. Experiments demonstrated the effectiveness of the proposed method. A case study indicated that the proposed method can predict the novel DTIs, which are not recorded on the original DTI dataset but are supported by recent studies.

7.2 Future Work

This final section provides several promising directions of the related future work.

- **Towards robust dynamic network embedding.**
  There are two straightforward approaches to generate a dynamic network. First, the dynamic network can be generated by slicing with a fixed number of edges [30, 31]. Second, the dynamic network can be generated by slicing with a fixed time interval [35, 59]. For the both cases, the degree of changes could be quite different even for the dynamic networks generated from the same dataset. In Chapter 5, we proposed a robust DNE for the first case [61], while a robust DNE for the second case was left for future work as suggested in our works [61, 63]. The second scenario is more challenging, because the number of changed edges may vary a lot over timesteps, e.g., much more friendships than usual would be built in a social network during a social event. The second scenario tries to study the performance of a DNE method at each timestep, while the first scenario focuses more on the average performance over all timesteps. Ideally, a desirable DNE method should be effective and robust to not only different degrees of changes by different slicing settings, but also different numbers of streaming edges over timesteps. Therefore, it is also worth investigating the second scenario.
  Apart from above two approaches to generate a dynamic network, there could exit more complicated approaches. It is impossible to list all. Consequently, another important future direction is to rethink of how to represent a dynamic network in a more proper way to reduce the uncertainties in generating dynamic networks, so that we can then propose a robust DNE method for broader applications [63].

- **Dynamic attributed network embedding for sparse topology in early stages.**
  Chapter 3 proposed a static attributed network embedding method [58], while Chapter 4 and 5 developed two dynamic plain network embedding methods [59, 61] respectively. It is natural to come up with combining both, i.e., to embed a dynamic attributed network. A dynamic attributed network is an attributed network with the evolution of both topologies and attributes over time. It can better model some real-world complex

systems, e.g., the relationships between social users and the profiles of social users would both vary over time. To our best knowledge, there are three dynamic attributed network embedding methods [27, 129, 183]. However, these methods have not considered the sparse topology of an dynamic attributed network in early stages (maybe called as a cold-start problem), which would affect the use of node attributes in a topological sparse network as discussed in Section 3.1. As a result, one future direction is to study the effect of sparse topology in early stages on dynamic attributed network embedding. Moreover, another interesting direction is to investigate the interplay between network topology and node attributes as a dynamic attributed network evolves.

- **Theoretical analysis or deeper insights of the proposed methods.**
  In this thesis, we proposed and empirically studied three novel NE methods, each of which was for addressing one set of the new challenges that we discovered from more realistic networks. Although extensive experiments were conducted to demosrate the superiorities and good properties of the proposed methods, the limitation of these proposed methods might be the lack of solid theoretical analysis or deeper insights. For example, we adopted softmax, i.e., Eq. (4.4) in Chapter 4, to normalize the score that quantifies the accumulated topological changes of a node in a sub-network snapshot of a dynamic network. The motivation of using softmax (often used in machine learning for multiclass classification problems) was simply to obtain a valid probability distribution such that a larger score gives a larger probability to select nodes largely affected by streaming edges. In fact, we may interpret the softmax here from the perspective of Boltzmann distribution in statistical physics, if an input dynamic network can be regarded as a complex system that follows thermodynamics. From this perspective, we may redesign the score function in Eq. (4.3) to include both an energy-like quantity and a temperature-like system constant in a more principle way, which might make the proposed method better motivated, enable researchers to conduct some theoretical analysis on it, and even develop a more effective method or new theory from it.

- **An end-to-end network embedding based DTI prediction method.**
  In Chapter 6, we introduced a NE based DTI prediction method [62], which consisted of

the feature extraction step and the DTI prediction step. Specifically, NE was applied to each input network to extract node features, which were then used to construct training samples, train a DTI prediction model, and make DTI predictions. It should be noted that, the node features were learned separately from the final prediction model. Therefore, the node features might not be optimal for the final prediction model. To fill this gap, we are now working on an end-to-end NE based DTI prediction method as listed in Section 1.4, titled as (might be slightly modified) "A Neural Embedding Model for Drug-Target Interaction Prediction with Micro and Macro Proximities Preservation" or GraphDTIP for short. For this ongoing work[1], the node features (or node embeddings) would be jointed optimized with the final prediction model.

- **Specific applications of dynamic network embedding.**

  There have been a large number of specific applications of static network embedding as surveyed in [48–53]. However, most real-world networks are dynamic by nature. Recently, the research interests of applications start moving to DNE. For instance, our DNE work [184] briefly presented in Appendix A.3 (i.e., an early version of our work [59] presented in Chapter 4) was adapted to dynamic vehicle trajectory clustering [54]. The specific DNE applications to real-world problems, which benefit business, scientific research, and our daily life, would further motivate and boost the development of DNE. Hence, it is needed to further explore specific DNE applications.

- **Beyond node level embeddings.**

  Considering the output of network embedding, this thesis mainly focused on learning node embeddings (i.e., node level embeddings or lowest level embeddings). However, some specific real-world applications require the higher (or coarser) level embeddings such as edge level, sub-network level, and whole network level [50]. One straightforward approach is to define a pooling or readout operation [11] to obtain the higher level embeddings based on the node level embeddings But this would neglect the direct semantic meaning between the higher level objects, as node level embeddings mainly preserve similarities between nodes. In the future, we may need to carefully consider

[1] https://github.com/mythezone/GraphDTIP

the direct semantic meaning between the higher level objects, rather than simply using a pooling or readout operation over node embeddings to obtain embeddings for the higher level objects, e.g., sub-network embeddings.

- **Input network enhancement.**

  In Chapter 3, we proposed an attributed network embedding method particularly for handling the topological sparse network [58]. To this end, node attributes were used to enrich the topological sparse network, as a kind of network enhancement mechanism, to improve the performance of its following Skip-Gram embedding model. Besides, there is also an interesting work about network enhancement to denoise weighted biological networks [185], though this work is neither directly related to network embedding nor related to attributed networks. Inspired by above two works, a novel angle to improve network embedding could be including an input network enhancement mechanism. It might be more intuitive to include an input network enhancement mechanism for attributed network embedding as we did in [58]. But the work [185] shows a way of network enhancement without using node attributes. It is promising for future work, from this novel angle, to improve network embedding by including an input network enhancement (or denoising) mechanism for various types of networks.

Appendix A

Supplementary Materials

A.1 A Proof of Eq. (3.1) in Chapter 3

In Eq. (3.1), we have

$$\mathbf{T} = \sum_{q=1}^{p} \lambda^{(q)} \mathbf{T}^{(q)} \quad \text{s.t.} \quad \sum_{q=1}^{p} \lambda^{(q)} = 1 \tag{A.1}$$

where the superscript $q \in \{1, ..., p\}$ denotes different sources of information and the factor $\lambda^{(q)} \in [0, 1]$ is a non-negative real number. Prove that $\mathbf{T}$ is a transition matrix given each $\mathbf{T}^{(q)}$ is a transition matrix[1].

According to the definition of a transition matrix, each row of $\mathbf{T}^{(q)}$ sums to one i.e.

$$\sum_{j=1}^{n} T_{ij}^{(q)} = 1 \tag{A.2}$$

where $T_{ij}^{(q)}$ is the entry (i, j) of the transition matrix at q source, which indicates the transition probability (or similarity) from node i to j; n is the number of nodes, and the subscript $i, j \in \{1, ..., n\}$. With Eq. (A.2) and the constraint in Eq. (A.1), it might be easy to show

[1]In mathematics, a transition matrix is also known as a stochastic matrix to describe the transitions of a Markov chain.

that each row of $\mathbf{T}$ sums to one as follows.

$$\begin{aligned}\sum_{j=1}^{n} T_{ij} &= \sum_{q=1}^{p}\sum_{j=1}^{n} \lambda^{(q)} T_{ij}^{(q)} \\ &= \sum_{q=1}^{p} \lambda^{(q)} (\sum_{j=1}^{n} T_{ij}^{(q)}) \\ &= \sum_{q=1}^{p} \lambda^{(q)} \cdot 1 \\ &= 1\end{aligned} \tag{A.3}$$

Note that, $\lambda^{(q)}$ is a scalar and thus can be multiplied with each entry of the matrix. Eq. (A.3) demosrates that each row of $\mathbf{T} \in \mathcal{R}^{n\times n}$ sums to one. By the definition of a transition matrix, $\mathbf{T}$ is hence a transition matrix.

A.2 A Proof of Eq. (3.3) in Chapter 3

For Eq. (3.3), we can rewrite as

$$\cos(\mathbf{x}, \mathbf{y}) = 1 - \frac{||\mathbf{x} - \mathbf{y}||^2}{2} \quad s.t. \quad ||\mathbf{x}|| = ||\mathbf{y}|| = 1 \tag{A.4}$$

where $||\cdot||$ denotes L2 norm of a vector; $||\mathbf{x} - \mathbf{y}||$ gives the Euclidean distance between $\mathbf{x}$ and $\mathbf{y}$; and $\cos(\mathbf{x}, \mathbf{y})$ represents their Cosine similarity. Prove that Eq. (A.4) holds for any pair of vectors $\mathbf{x} \neq \mathbf{0}$ and/or $\mathbf{y} \neq \mathbf{0}$.

According to the definition of Euclidean distance, we have:

$$\begin{aligned}\text{Euclidean}(\mathbf{x}, \mathbf{y}) = ||\mathbf{x} - \mathbf{y}|| &= \left[\sum_i (x_i - y_i)^2\right]^{\frac{1}{2}} \\ &= \left[\sum_i x_i^2 + y_i^2 - 2x_i y_i\right]^{\frac{1}{2}} \\ &= \left[\sum_i x_i^2 + \sum_i y_i^2 - 2\sum_i x_i y_i\right]^{\frac{1}{2}} \\ &= \left[\mathbf{x}^{\mathrm{T}}\mathbf{x} + \mathbf{y}^{\mathrm{T}}\mathbf{y} - 2\mathbf{x}^{\mathrm{T}}\mathbf{y}\right]^{\frac{1}{2}}\end{aligned} \tag{A.5}$$

According to the definition of Cosine similarity, we have:

$$\cos(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x}^{\mathrm{T}}\mathbf{y}}{||\mathbf{x}||||\mathbf{y}||} \tag{A.6}$$

With the constant in Eq. (A.4) that $||\mathbf{x}|| = ||\mathbf{y}|| = 1$ i.e. projecting $\mathbf{x}$ and $\mathbf{y}$ into a unit hyper-sphere, we have:

$$\begin{aligned} \text{Euclidean}(\mathbf{x}, \mathbf{y}) &= \left[2 - 2\mathbf{x}^{\mathrm{T}}\mathbf{y}\right]^{\frac{1}{2}} \\ &= \left[2 - 2\cos(\mathbf{x}, \mathbf{y})\right]^{\frac{1}{2}} \end{aligned} \tag{A.7}$$

By rearranging Eq. (A.7), it is easy to show that:

$$\begin{aligned} \cos(\mathbf{x}, \mathbf{y}) &= \frac{2 - \text{Euclidean}(\mathbf{x}, \mathbf{y})^2}{2} \\ &= 1 - \frac{||\mathbf{x} - \mathbf{y}||^2}{2} \end{aligned} \tag{A.8}$$

A.3 An Early Version of GloDyNE in Chapter 4

In this section, we first give the abstract of our work called DynWalks [184], which is an early version of our work called GloDyNE [59] in chapter 4. After that, the main different between DynWalks and GloDyNE is presented. And finally, we refer the readers who are interested in DynWalks to the original manuscript. Note that, the contents below are mainly adapted from our work [184].

Learning topological representation of a network in dynamic environments has recently attracted considerable attention due to the time-evolving nature of many real-world networks i.e. nodes/links might be added/removed as time goes on. Dynamic network embedding aims to learn low dimensional embeddings for unseen and seen nodes by using any currently available snapshots of a dynamic network. For seen nodes, the existing methods either treat them equally important or focus on the k most affected nodes at each time step. However, the former solution is time-consuming, and the later solution that relies on incoming changes may lose the global topology—an important feature for downstream tasks. To address these challenges, we propose a dynamic network embedding method called DynWalks, which includes two key components: 1) An online network embedding framework that can

dynamically and efficiently learn embeddings based on the selected nodes; 2) A novel online node selecting scheme that offers the flexible choices to balance global topology and recent changes, as well as to fulfill the real-time constraint if needed. The empirical studies on six real-world dynamic networks under three different slicing ways show that DynWalks significantly outperforms the state-of-the-art methods in graph reconstruction tasks, and obtains comparable results in link prediction tasks. Furthermore, the wall-clock time and complexity analysis demonstrate its excellent time and space efficiency.

The main different between DynWalks and GloDyNE is the node selecting scheme. As a result, here we present the idea of (online) node selecting scheme used in DynWalks as shown in Figure A.1.

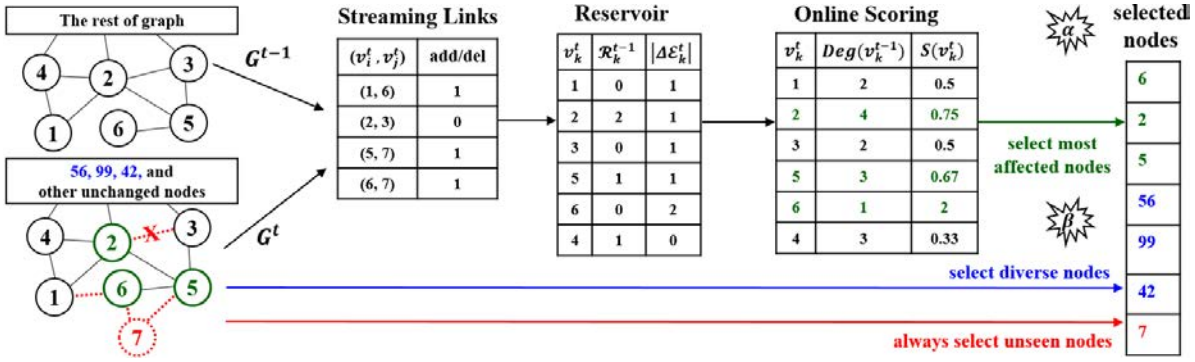


Figure A.1: The illustration of the proposed online node selecting scheme in DynWalks.

The online node selecting scheme is proposed to select nodes from current snapshot G^t. There are two questions to be answered. Firstly, how many nodes should be selected? To reduce time and space complexity, one natural idea is to focus on a part of important nodes. Therefore, the hyper-parameter α is used to limit the number of selected nodes to $\alpha|\mathcal{V}^t|$ where α can be adapted according to the real-time constraint if needed. And secondly, which nodes should be selected? To balance recent changes and global topology, the hyper-parameter β is used to select $\beta\alpha|\mathcal{V}_{\text{all}}^t|$ most affected nodes and $(1-\beta)\alpha|\mathcal{V}_{\text{all}}^t|$ diverse nodes.

The original manuscript of DynWalks, with the details of how to select most affected nodes and diverse nodes, are available at `https://arxiv.org/abs/1907.11968`. Note that, DynWalks has been applied to dynamic vehicle trajectory clustering [54], which demosrates its real-world significance.

A.4 Additional Results of Ablation Study for Chapter 5

In addition to Table 5.3 (for GR tasks), we provide the results of ablation study for NR and LP tasks as shown in Table A.1.

Table A.1: Ablation study of SG-EDNE for NR and LP tasks.

	DNC-Email	College-Msg	Co-Author	FB-Wall	Wiki-Talk
NR-MAP@5					
DNE-rw	81.22±0.71	88.98±1.85	96.00±0.99	94.41±1.02	**92.62±1.52**
EDNE-rw-fix	87.90±1.10	87.46±0.59	97.18±0.43	93.56±0.67	70.97±1.11
EDNE-rw	**94.24±1.33**	92.34±0.21	98.67±0.15	**95.82±0.24**	85.53±0.71
EDNE-rwr	**95.02±0.97**	**92.80±0.20**	**98.94±0.09**	**96.60±0.20**	87.18±0.78
EDNE-rwr-ws	93.62±0.85	**94.44±0.19**	**98.69±0.18**	95.09±0.38	**87.64±0.79**
NR-MAP@50					
DNE-rw	67.46±0.59	71.15±2.88	87.43±1.30	86.72±1.87	**81.14±3.35**
EDNE-rw-fix	74.62±1.22	67.12±1.20	87.70±0.60	82.10±1.22	54.54±0.71
EDNE-rw	**82.58±1.85**	**75.60±0.19**	**90.81±0.13**	**86.13±0.29**	68.61±0.29
EDNE-rwr	**83.68±1.60**	**76.50±0.23**	**91.35±0.11**	**86.84±0.28**	70.63±0.49
EDNE-rwr-ws	80.53±1.28	74.93±0.29	89.32±0.30	82.89±0.52	**71.33±0.66**
LP-AUC-L1-feature					
DNE-rw	84.86±0.60	72.49±0.34	87.91±0.33	84.35±0.49	77.14±0.94
EDNE-rw-fix	86.21±0.44	73.35±0.41	90.43±0.54	87.29±0.11	85.16±0.58
EDNE-rw	**87.48±0.64**	**74.79±0.50**	**90.90±0.68**	**87.83±0.02**	**86.29±0.68**
EDNE-rwr	**87.47±0.63**	**74.44±0.39**	**90.92±0.70**	**88.01±0.08**	**86.28±0.61**
EDNE-rwr-ws	85.16±1.11	70.28±0.25	89.36±0.23	84.61±1.37	84.96±0.49
LP-AUC-L2-feature					
DNE-rw	85.96±0.48	75.01±0.40	89.17±0.41	85.69±0.27	78.79±0.66
EDNE-rw-fix	87.13±0.32	75.48±0.47	91.34±0.49	87.92±0.07	86.13±0.57
EDNE-rw	**88.32±0.53**	**77.01±0.58**	**91.89±0.58**	**88.45±0.03**	**87.37±0.66**
EDNE-rwr	**88.34±0.49**	**76.63±0.44**	**91.93±0.61**	**88.67±0.05**	**87.34±0.58**
EDNE-rwr-ws	85.14±1.00	70.29±0.27	89.54±0.20	83.24±2.17	85.31±0.51

The findings for NR and LP tasks are similar to that for GR tasks as discussed in Section 5.4.2.2. There are two exceptions. First, EDNE-rwr-ws outperforms EDNE-rwr on College-Msg in NR-MAP@5 and on Wiki-Talk in both NR tasks. This finding indicates that the scaling operation, i.e., [0, 1] min-max scaling, may fail sometimes. However, for all other 27/30 cases, the scaling operation can improve the performance.

Second, DNE-rw (without ensembles) obtains the worst results than all other variants of DNE ensembles in LP tasks, while DNE-rw obtains the best results in NR tasks GR tasks (see Table 5.3). This observation may encourage us to employ the ensembles in LP tasks.

A.5 Additional Results of Parameter Sensitivity for Chapter 5

In addition to Figure 5.7 (for GR tasks), we provide the results of parameter sensitivity for NR and LP tasks as shown in Figure A.2 and A.3. The findings for NR and LP tasks are the same to that for GR tasks as discussed in Section 5.4.2.3.

One new observation, according to Figure A.3, is that LP tasks often prefer a lager M than GR and NR tasks. This observation again encourages us to employ the ensembles in LP tasks as also suggested in Appendix A.4 above.

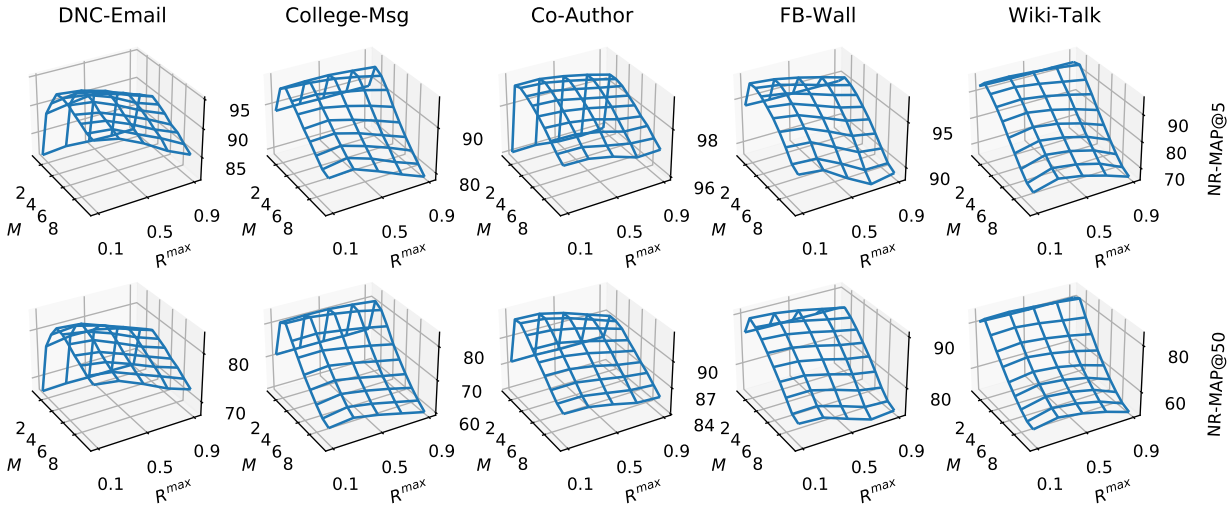


Figure A.2: Parameter sensitivity for NR tasks. M is for the number of base models and R^{max} is for the maximum restart probability.

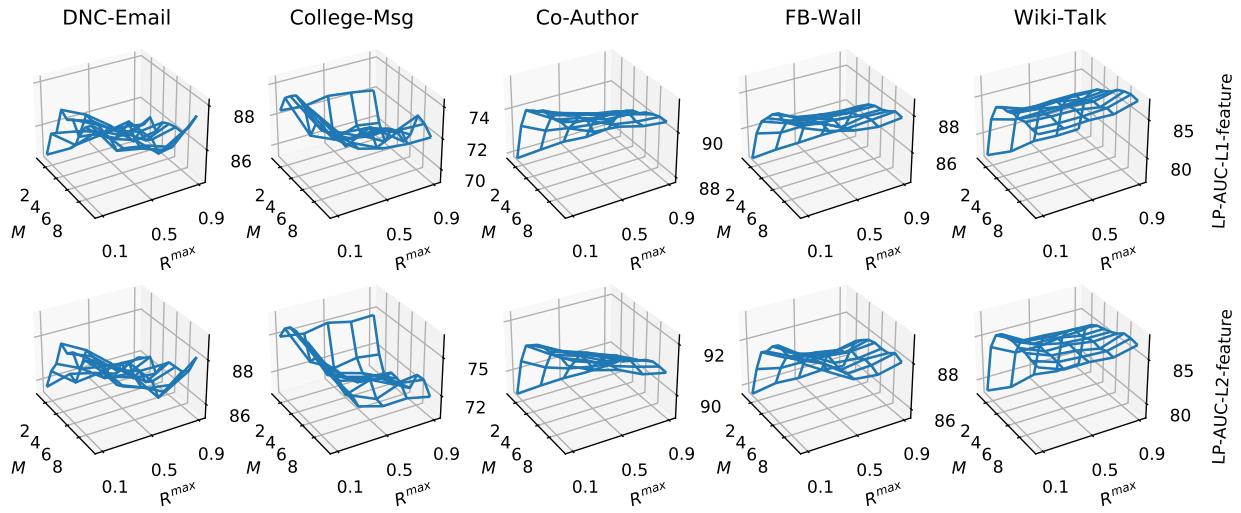


Figure A.3: Parameter sensitivity for LP tasks. M is for the number of base models and R^{max} is for the maximum restart probability.

Appendix B

Useful Resources

Source Code for The Thesis.

- To reproduce RoSANE in chapter 3: `https://github.com/houchengbin/OpenANE`
- To reproduce GloDyNE in chapter 4: `https://github.com/houchengbin/GloDyNE`
- To reproduce SG-EDNE in chapter 5: `https://github.com/houchengbin/SG-EDNE`
- To reproduce NE-DTIP in chapter 6: `https://github.com/BrisksHan/NE-DTIP`
- To reproduce HSRL as mentioned in chapter 3: `https://github.com/fuguoji/HSRL`
- To reproduce DynWalks as mentioned in Appendix A.3 (an early version of GloDyNE): `https://github.com/houchengbin/GloDyNE/releases/tag/v0.1`
- To reproduce GraphDTIP as mentioned in chapter 7 (ongoing work): `https://github.com/mythezone/GraphDTIP`

Network Datasets.

- Prepared data for the thesis: `https://github.com/houchengbin/NetEmb-Datasets`
- Stanford large network dataset collection: `https://snap.stanford.edu/data`
- Dynamic networks: `http://networkrepository.com/dynamic.php`
- KONECT Project: `http://konect.cc`
- And others:

https://www.aminer.cn/data

https://linqs.soe.ucsc.edu/data

http://networkrepository.com/index.php

http://snap.stanford.edu/biodata/index.html

http://vlado.fmf.uni-lj.si/pub/networks/data

https://sites.google.com/site/ucinetsoftware/datasets

https://cnets.indiana.edu/data-repository-for-nan-group

http://networksciencebook.com/translations/en/resources/data.html

Others.

- A list of NE papers: https://github.com/thunlp/NRLPapers
- A list of DNE papers: https://github.com/Cantoria/dynamic-graph-papers
- A list of GNN papers: https://github.com/thunlp/GNNPapers
- Source code of a list of NE methods: https://github.com/thunlp/OpenNE
- Source code of a list of ANE methods: https://github.com/houchengbin/OpenANE
- NE or GNN framework: https://github.com/alibaba/graph-learn
- NE or GNN framework https://github.com/facebookresearch/PyTorch-BigGraph
- NE or GNN framework: https://github.com/dmlc/dgl
- NE or GNN framework: https://github.com/PaddlePaddle/PGL
- NE or GNN framework: https://github.com/rusty1s/pytorch_geometric
- Python package for network science: https://networkx.org
- Python package for machine learning: https://scikit-learn.org/stable
- Python package for Skip-Gram: https://github.com/RaRe-Technologies/gensim

References

[1] M. Hilbert, "Big data for development: A review of promises and challenges," *Development Policy Review*, vol. 34, no. 1, pp. 135–174, 2016.

[2] Domo Inc., *Data never sleeps 8.0*, `https://www.domo.com/learn/data-never-sleeps-8`, Accessed: 2021-05-18, 2021.

[3] Z. Wang, J. Liao, Q. Cao, H. Qi, and Z. Wang, "Friendbook: A semantic-based friend recommendation system for social networks," *IEEE transactions on mobile computing*, vol. 14, no. 3, pp. 538–551, 2015.

[4] S. Bhagat, G. Cormode, and S. Muthukrishnan, "Node classification in social networks," in *Social network data analytics*, Springer, 2011, pp. 115–148.

[5] M. McPherson, L. Smith-Lovin, and J. M. Cook, "Birds of a feather: Homophily in social networks," *Annual Review of Sociology*, vol. 27, no. 1, pp. 415–444, 2001.

[6] O. Tsur and A. Rappoport, "What's in a hashtag?: Content based prediction of the spread of ideas in microblogging communities," in *WSDM*, ACM, 2012, pp. 643–652.

[7] A. L. Hu and K. C. Chan, "Utilizing both topological and attribute information for protein complex identification in ppi networks," *IEEE/ACM transactions on computational biology and bioinformatics*, vol. 10, no. 3, pp. 780–792, 2013.

[8] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[9] L. Cayton, "Algorithms for manifold learning," *Univ. of California at San Diego Tech. Rep*, vol. 12, no. 1-17, p. 1, 2005.

[10] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[11] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, 2020.

[12] M. Verleysen and D. François, "The curse of dimensionality in data mining and time series prediction," in *International Work-Conference on Artificial Neural Networks*, Springer, 2005, pp. 758–770.

[13] M. Bagherian, E. Sabeti, K. Wang, M. A. Sartor, Z. Nikolovska-Coleska, and K. Najarian, "Machine learning approaches and databases for prediction of drug–target interaction: A survey paper," *Briefings in bioinformatics*, vol. 22, no. 1, pp. 247–269, 2021.

[14] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Chang, "Network representation learning with rich text information," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[15] S. Pan, J. Wu, X. Zhu, C. Zhang, and Y. Wang, "Tri-party deep network representation," in *IJCAI*, 2016, pp. 1895–1901.

[16] X. Huang, J. Li, and X. Hu, "Accelerated attributed network embedding," in *Proceedings of the 2017 SIAM international conference on data mining*, SIAM, 2017, pp. 633–641.

[17] L. Liao, X. He, H. Zhang, and T.-S. Chua, "Attributed social network embedding," *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[18] H. Gao and H. Huang, "Deep attributed network embedding," in *IJCAI*, 2018, pp. 3364–3370.

[19] W. Liu, Z. Liu, F. Yu, P.-y. Chen, T. Suzumura, and G. Hu, "A scalable attribute-aware network embedding system," *Neurocomputing*, vol. 339, pp. 279–291, 2019.

[20] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NIPS*, 2017, pp. 1024–1034.

[21] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.

[22] H. Wang, E. Chen, Q. Liu, T. Xu, D. Du, W. Su, and X. Zhang, "A united approach to learning sparse attributed network embedding," in *2018 IEEE International Conference on Data Mining (ICDM)*, IEEE, 2018, pp. 557–566.

[23] J. Nešetřil and P. O. De Mendez, *Sparsity: graphs, structures, and algorithms*. Springer Science & Business Media, 2012, vol. 28.

[24] M. Hutson, "Artificial intelligence faces reproducibility crisis," *Science*, vol. 359, no. 6377, pp. 725–726, 2018, ISSN: 0036-8075. DOI: 10.1126/science.359.6377.725. eprint: http://science.sciencemag.org/content/359/6377/725.full.pdf. [Online]. Available: http://science.sciencemag.org/content/359/6377/725.

[25] K. Tu, J. Ma, P. Cui, J. Pei, and W. Zhu, "Autone: Hyperparameter optimization for massive network embedding," in *KDD*, 2019.

[26] L. Zhu, D. Guo, J. Yin, G. Ver Steeg, and A. Galstyan, "Scalable temporal latent space inference for link prediction in dynamic social networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 10, pp. 2765–2777, 2016.

[27] J. Li, H. Dani, X. Hu, J. Tang, Y. Chang, and H. Liu, "Attributed network embedding for learning in a dynamic environment," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, ACM, 2017, pp. 387–396.

[28] P. Goyal, N. Kamra, X. He, and Y. Liu, "Dyngem: Deep embedding method for dynamic graphs," in *IJCAI International Workshop on Representation Learning for Graphs*, 2017.

[29] D. Zhu, P. Cui, Z. Zhang, J. Pei, and W. Zhu, "High-order proximity preserved embedding for dynamic networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 11, pp. 2134–2144, 2018.

[30] Z. Zhang, P. Cui, J. Pei, X. Wang, and W. Zhu, "Timers: Error-bounded svd restart on dynamic networks," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[31] L. Du, Y. Wang, G. Song, Z. Lu, and J. Wang, "Dynamic network embedding: An extended approach for skip-gram based network embedding.," in *IJCAI*, 2018, pp. 2086–2092.

[32] L. Zhou, Y. Yang, X. Ren, F. Wu, and Y. Zhuang, "Dynamic network embedding by modeling triadic closure process," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[33] X. Chen, P. Cui, L. Yi, and S. Yang, "Scalable optimization for embedding highly-dynamic and recency-sensitive data," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ACM, 2018, pp. 130–138.

[34] S. Mahdavi, S. Khoshraftar, and A. An, "Dynnode2vec: Scalable dynamic network embedding," in *IEEE International Conference on Big Data (Big Data)*, 2018, pp. 3762–3765.

[35] U. Singer, I. Guy, and K. Radinsky, "Node embedding over temporal graphs," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

[36] R. Trivedi, M. Farajtabar, P. Biswal, and H. Zha, "Dyrep: Learning representations over dynamic graphs," in *International Conference on Learning Representations*, 2019.

[37] B. Yu, B. Lu, C. Zhang, C. Li, and K. Pan, "Node proximity preserved dynamic network embedding via matrix perturbation," *Knowl. Based Syst.*, vol. 196, p. 105 822, 2020.

[38] P. Goyal, S. R. Chhetri, and A. Canedo, "Dyngraph2vec: Capturing network dynamics using dynamic graph representation learning," *Knowl. Based Syst.*, vol. 187, 2020.

[39] L. Ma, Y. Zhang, J. Li, Q. Lin, Q. Bao, S. Wang, and M. Gong, "Community-aware dynamic network embedding by using deep autoencoder," *Inf. Sci.*, vol. 519, pp. 22–42, 2020.

[40] W. Yu, W. Cheng, C. C. Aggarwal, K. Zhang, H. Chen, and W. 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 & Data Mining*, ACM, 2018, pp. 2672–2681.

[41] H. Peng, J. Li, H. Yan, Q. Gong, S. Wang, L. Liu, L. Wang, and X. Ren, "Dynamic network embedding via incremental skip-gram with negative sampling," *Sci. China Inf. Sci.*, vol. 63, no. 10, pp. 1–19, 2020.

[42] H. P. Sajjad, A. Docherty, and Y. Tyshetskiy, "Efficient representation learning using random walks for dynamic graphs," *CoRR*, vol. abs/1901.01346, 2019.

[43] L. Zheng, Z. Li, J. Li, Z. Li, and J. Gao, "Addgraph: Anomaly detection in dynamic graph using attention-based temporal gcn," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, AAAI Press, 2019, pp. 4419–4425.

[44] A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, T. B. Schardl, and C. E. Leiserson, "Evolvegcn: Evolving graph convolutional networks for dynamic graphs.," in *AAAI*, 2020, pp. 5363–5370.

[45] D. Xu, C. Ruan, E. Korpeoglu, S. Kumar, and K. Achan, "Inductive representation learning on temporal graphs," in *ICLR*, 2020.

[46] M. Gong, S. Ji, Y. Xie, Y. Gao, and A. Qin, "Exploring temporal information for dynamic network embedding," *IEEE Trans. Knowl. Data Eng.*, 2020.

[47] L. Qu, H. Zhu, Q. Duan, and Y. Shi, "Continuous-time link prediction via temporal dependent graph neural network," in *Proceedings of The Web Conference 2020*, 2020, pp. 3026–3032.

[48] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *IEEE Data Engineering Bulletin*, vol. 40, no. 3, pp. 52–74, 2017.

[49] P. Cui, X. Wang, J. Pei, and W. Zhu, "A survey on network embedding," *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[50] H. Cai, V. W. Zheng, and K. Chang, "A comprehensive survey of graph embedding: Problems, techniques and applications," *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[51] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowledge-Based Systems*, vol. 151, pp. 78–94, 2018.

[52] H. Chen, B. Perozzi, R. Al-Rfou, and S. Skiena, "A tutorial on network embeddings," *arXiv preprint arXiv:1808.02590*, 2018.

[53] D. Zhang, J. Yin, X. Zhu, and C. Zhang, "Network representation learning: A survey," *IEEE Trans. Big Data*, vol. 6, no. 1, pp. 3–28, 2020.

[54] W. Wang, F. Xia, H. Nie, Z. Chen, Z. Gong, X. Kong, and W. Wei, "Vehicle trajectory clustering based on dynamic representation learning of internet of vehicles," *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[55] W. Jiang and J. Luo, "Graph neural network for traffic forecasting: A survey," *arXiv preprint arXiv:2101.11174*, 2021.

[56] Y. Luo, X. Zhao, J. Zhou, J. Yang, Y. Zhang, W. Kuang, J. Peng, L. Chen, and J. Zeng, "A network integration approach for drug-target interaction prediction and computational drug repositioning from heterogeneous information," *Nature communications*, vol. 8, no. 1, pp. 1–13, 2017.

[57] M. A. Thafar, R. S. Olayan, H. Ashoor, S. Albaradei, V. B. Bajic, X. Gao, T. Gojobori, and M. Essack, "Dtigems+: Drug–target interaction prediction using graph embedding, graph mining, and similarity-based techniques," *Journal of Cheminformatics*, vol. 12, no. 1, pp. 1–17, 2020.

[58] C. Hou, S. He, and K. Tang, "RoSANE: Robust and Scalable Attributed Network Embedding for Sparse Networks," *Neurocomputing*, 2020. DOI: 10.1016/j.neucom.2020.05.080.

[59] C. Hou, H. Zhang, S. He, and K. Tang, "GloDyNE: Global topology preserving dynamic network embedding," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2020. DOI: 10.1109/TKDE.2020.3046511.

[60] ——, "GloDyNE: Global topology preserving dynamic network embedding (extended abstract)," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, 2022.

[61] C. Hou, G. Fu, P. Yang, Z. Hu, S. He, and K. Tang, "Robust dynamic network embedding via ensembles," *arXiv preprint arXiv:2105.14557*, 2021.

[62] H. Zhang, C. Hou, D. McDonald, and S. He, "A network embedding based approach to drug-target interaction prediction using additional implicit networks," in *International Conference on Artificial Neural Networks (ICANN)*, 2021, pp. 491–503. DOI: 10.1007/978-3-030-86362-3_40.

[63] C. Hou and K. Tang, "Towards robust dynamic network embedding," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2021. DOI: 10.24963/ijcai.2021/676.

[64] G. Fu, C. Hou, and X. Yao, "Learning topological representation for networks via hierarchical sampling," in *2019 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2019, pp. 1–8. DOI: 10.1109/IJCNN.2019.8851893.

[65] J. B. Tenenbaum, V. De Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[66] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering," in *Advances in neural information processing systems*, 2002, pp. 585–591.

[67] S. Cao, W. Lu, and Q. Xu, "Grarep: Learning graph representations with global structural information," in *CIKM*, Melbourne, Australia: ACM, 2015, pp. 891–900, ISBN: 978-1-4503-3794-6. DOI: 10.1145/2806416.2806512. [Online]. Available: http://doi.acm.org/10.1145/2806416.2806512.

[68] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2016, pp. 1105–1114.

[69] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *NIPS*, 2013, pp. 3111–3119.

[70] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.

[71] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[72] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017.

[73] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *KDD*, ACM, 2014, pp. 701–710.

[74] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *KDD*, ACM, 2016, pp. 855–864.

[75] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[76] B. C. Csáji, "Approximation with artificial neural networks," *Faculty of Sciences, Etvs Lornd University, Hungary*, vol. 24, p. 48, 2001.

[77] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2016, pp. 1225–1234.

[78] S. Cao, W. Lu, and Q. Xu, "Deep neural networks for learning graph representations," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[79] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in neural information processing systems*, 2016, pp. 3844–3852.

[80] Y.-A. Lai, C.-C. Hsu, W. H. Chen, M.-Y. Yeh, and S.-D. Lin, "Preserving proximity and global ranking for node embedding," in *Advances in Neural Information Processing Systems*, 2017, pp. 5261–5270.

[81] M. A. Kramer, "Nonlinear principal component analysis using autoassociative neural networks," *AIChE journal*, vol. 37, no. 2, pp. 233–243, 1991.

[82] A. Ortega, P. Frossard, J. Kovačević, J. M. Moura, and P. Vandergheynst, "Graph signal processing: Overview, challenges, and applications," *Proceedings of the IEEE*, vol. 106, no. 5, pp. 808–828, 2018.

[83] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020.

[84] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[85] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[86] A. Lerer, L. Wu, J. Shen, T. Lacroix, L. Wehrstedt, A. Bose, and A. Peysakhovich, "PyTorch-BigGraph: A Large-scale Graph Embedding System," in *Proceedings of the 2nd SysML Conference*, Palo Alto, CA, USA, 2019.

[87] W. Huang, T. Zhang, Y. Rong, and J. Huang, "Adaptive sampling towards fast graph representation learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 4558–4567.

[88] I. Chami, S. Abu-El-Haija, B. Perozzi, C. Ré, and K. Murphy, "Machine learning on graphs: A model and comprehensive taxonomy," *arXiv preprint arXiv:2005.03675*, 2020.

[89] X. Huang, J. Zhang, D. Li, and P. Li, "Knowledge graph embedding based question answering," in *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, 2019, pp. 105–113.

[90] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, and Z. Zhang, "Deep graph library: A graph-centric, highly-performant package for graph neural networks," *arXiv preprint arXiv:1909.01315*, 2019.

[91] R. Zhu, K. Zhao, H. Yang, W. Lin, C. Zhou, B. Ai, Y. Li, and J. Zhou, "Aligraph: A comprehensive graph neural network platform," *Proceedings of the VLDB Endowment*, vol. 12, no. 12, pp. 2094–2105, 2019.

[92] P. Holme, "Modern temporal network theory: A colloquium," *The European Physical Journal B*, vol. 88, no. 9, p. 234, 2015.

[93] M. Latapy, T. Viard, and C. Magnien, "Stream graphs and link streams for the modeling of interactions over time," *Social Network Analysis and Mining*, vol. 8, no. 1, p. 61, 2018.

[94] S. M. Kazemi, R. Goel, K. Jain, I. Kobyzev, A. Sethi, P. Forsyth, and P. Poupart, "Representation learning for dynamic graphs: A survey," *J. Mach. Learn. Res.*, vol. 21, 70:1–70:73, 2020.

[95] C. D. Barros, M. R. Mendonça, A. B. Vieira, and A. Ziviani, "A survey on embedding dynamic graphs," *arXiv preprint arXiv:2101.01229*, 2021.

[96] Y. Xie, C. Li, B. Yu, C. Zhang, and Z. Tang, "A survey on dynamic network embedding," *arXiv preprint arXiv:2006.08093*, 2020.

[97] J. Skardinga, B. Gabrys, and K. Musial, "Foundations and modelling of dynamic networks using dynamic graph neural networks: A survey," *IEEE Access*, 2021.

[98] G. Xue, M. Zhong, J. Li, J. Chen, C. Zhai, and R. Kong, "Dynamic network embedding survey," *arXiv preprint arXiv:2103.15447*, 2021.

[99] L. Lü and T. Zhou, "Link prediction in complex networks: A survey," *Physica A: Statistical Mechanics and its Applications*, vol. 390, no. 6, pp. 1150–1170, 2011.

[100] X. Wei, L. Xu, B. Cao, and P. S. Yu, "Cross view link prediction by learning noise-resilient representation consensus," in *WWW*, Perth, Australia: International World Wide Web Conferences Steering Committee, 2017, pp. 1611–1619, ISBN: 978-1-4503-4913-0. DOI: 10.1145/3038912.3052575. [Online]. Available: https://doi.org/10.1145/3038912.3052575.

[101] P. Kazienko and T. Kajdanowicz, "Label-dependent node classification in the network," *Neurocomputing*, vol. 75, no. 1, pp. 199–209, 2012.

[102] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *WWW*, 2015, pp. 1067–1077.

[103] S. Goswami, C. Murthy, and A. K. Das, "Sparsity measure of a network graph: Gini index," *Information Sciences*, vol. 462, pp. 16–39, 2018.

[104] J. Kunegis, "Konect: The koblenz network collection," in *Proceedings of the 22nd International Conference on World Wide Web*, ACM, 2013, pp. 1343–1350.

[105] D. Yu, R. Zhang, Z. Jiang, Y. Wu, and Y. Yang, "Graph-revised convolutional network," *arXiv preprint arXiv:1911.07123*, 2019.

[106] A. K. Bhowmick, K. Meneni, and B. Mitra, "On the network embedding in sparse signed networks," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Springer, 2019, pp. 94–106.

[107] R. Hong, Y. He, L. Wu, Y. Ge, and X. Wu, "Deep attributed network embedding by preserving structure and attribute information," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2019.

[108] D. McDonald and S. He, "Heat: Hyperbolic embedding of attributed networks," in *International Conference on Intelligent Data Engineering and Automated Learning*, Springer, 2020, pp. 28–40.

[109] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, "Community preserving network embedding.," in *AAAI*, 2017, pp. 203–209.

[110] W. Shi, L. Huang, C.-D. Wang, J.-H. Li, Y. Tang, and C. Fu, "Network embedding via community based variational autoencoder," *IEEE Access*, vol. 7, pp. 25 323–25 333, 2019.

[111] J.-H. Li, C.-D. Wang, L. Huang, D. Huang, J.-H. Lai, and P. Chen, "Attributed network embedding with micro-meso structure," in *International Conference on Database Systems for Advanced Applications*, Springer, 2018, pp. 20–36.

[112] Z. Li, X. Wang, J. Li, and Q. Zhang, "Structural role enhanced attributed network embedding," in *International Conference on Web Information Systems Engineering*, Springer, 2019, pp. 568–582.

[113] K. Wang, L. Xu, L. Huang, C.-D. Wang, Y. Tang, and C. Fu, "Inter-intra information preserving attributed network embedding," *IEEE Access*, vol. 7, pp. 79 463–79 476, 2019.

[114] S. Bandyopadhyay, S. V. Vivek, and M. Murty, "Outlier resistant unsupervised deep architectures for attributed network embedding," in *Proceedings of the 13th International Conference on Web Search and Data Mining*, 2020, pp. 25–33.

[115] B. Yu, Y. Li, C. Zhang, K. Pan, and Y. Xie, "Enhancing attributed network embedding via similarity measure," *IEEE Access*, vol. 7, pp. 166 235–166 245, 2019.

[116] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer networks and ISDN systems*, vol. 30, no. 1-7, pp. 107–117, 1998.

[117] T. Liu, A. W. Moore, and A. Gray, "New algorithms for efficient high-dimensional nonparametric classification," *Journal of Machine Learning Research*, vol. 7, no. Jun, pp. 1135–1158, 2006.

[118] S. M. Omohundro, *Five balltree construction algorithms*. International Computer Science Institute Berkeley, 1989.

[119] A. Strehl, J. Ghosh, and R. Mooney, "Impact of similarity measures on web-page clustering," in *Workshop on artificial intelligence for web search (AAAI 2000)*, vol. 58, 2000, p. 64.

[120] Y. Goldberg and O. Levy, "Word2vec explained: Deriving mikolov et al.'s negative-sampling word-embedding method," *arXiv preprint arXiv:1402.3722*, 2014.

[121] O. Levy and Y. Goldberg, "Neural word embedding as implicit matrix factorization," in *Advances in neural information processing systems*, 2014, pp. 2177–2185.

[122] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," English, in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, `http://is.muni.cz/publication/884893/en`, Valletta, Malta: ELRA, May 2010, pp. 45–50.

[123] H. Munaga and V. Jarugumalli, "Performance evaluation: Ball-treeand kd-tree in the context of mst," in *International Joint Conference on Advances in Signal Processing and Information Technology*, Springer, 2011, pp. 225–228.

[124] A. L. Traud, P. J. Mucha, and M. A. Porter, "Social structure of facebook networks," *Physica A: Statistical Mechanics and its Applications*, vol. 391, no. 16, pp. 4165–4180, 2012.

[125] Z. Yang, W. W. Cohen, and R. Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," *arXiv preprint arXiv:1603.08861*, 2016.

[126] M. G. Vozalis and K. G. Margaritis, "Applying svd on item-based filtering," in *5th International Conference on Intelligent Systems Design and Applications (ISDA'05)*, IEEE, 2005, pp. 464–469.

[127] V. Batagelj and U. Brandes, "Efficient generation of large random networks," *Physical Review E*, vol. 71, no. 3, p. 036 113, 2005.

[128] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2008.

[129] H. Wei, G. Hu, W. Bai, S. Xia, and Z. Pan, "Lifelong representation learning in dynamic attributed networks," *Neurocomputing*, vol. 358, pp. 1–9, 2019.

[130] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[131] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz, "Recent advances in graph partitioning," in *Algorithm Engineering*, Springer, 2016, pp. 117–158.

[132] Z. Zhang, P. Cui, X. Wang, J. Pei, X. Yao, and W. Zhu, "Arbitrary-order proximity preserved network embedding," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ACM, 2018, pp. 2778–2786.

[133] M. U. Gutmann and A. Hyvärinen, "Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics.," *Journal of Machine Learning Research*, vol. 13, no. 2, 2012.

[134] A. Mnih and Y. W. Teh, "A fast and simple algorithm for training neural probabilistic language models," in *Proceedings of the 29th International Coference on International Conference on Machine Learning*, 2012, pp. 419–426.

[135] P. Cui, X. Wang, J. Pei, and W. Zhu, "A survey on network embedding," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 5, pp. 833–852, 2019.

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

[137] G. H. Nguyen, J. B. Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim, "Continuous-time dynamic network embeddings," in *Companion Proceedings of the The Web Conference 2018*, 2018, pp. 969–976.

[138] M. Shi, Y. Tang, and X. Zhu, "Mlne: Multi-label network embedding," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 9, pp. 3682–3695, 2019.

[139] Y. Sun, K. Tang, Z. Zhu, and X. Yao, "Concept drift adaptation by exploiting historical knowledge," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 10, pp. 4822–4832, 2018.

[140] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, and M. Woniak, "Ensemble learning for data stream analysis: A survey," *Information Fusion*, vol. 37, pp. 132–156, 2017.

[141] V. Losing, B. Hammer, and H. Wersing, "Incremental on-line learning: A review and comparison of state of the art algorithms," *Neurocomputing*, vol. 275, pp. 1261–1274, 2018.

[142] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM computing surveys (CSUR)*, vol. 46, no. 4, pp. 1–37, 2014.

[143] G. Brown, J. Wyatt, R. Harris, and X. Yao, "Diversity creation methods: A survey and categorisation," *Information Fusion*, vol. 6, no. 1, pp. 5–20, 2005.

[144] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015. [Online]. Available: `http://networkrepository.com`.

[145] J. Leskovec and A. Krevl, *SNAP Datasets: Stanford large network dataset collection*, `http://snap.stanford.edu/data`, Jun. 2014.

[146] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[147] A. Kemper, *Valuation of network effects in software markets: A complex networks approach*. Springer Science & Business Media, 2009.

[148] S. Wang, H. Cho, C. Zhai, B. Berger, and J. Peng, "Exploiting ontology graph for predicting sparsely annotated gene function," *Bioinformatics*, vol. 31, no. 12, pp. i357–i364, 2015.

[149] E. W. Huang, S. Wang, and C. Zhai, "Visage: Integrating external knowledge into electronic medical record visualization.," in *PSB*, World Scientific, 2018, pp. 578–589.

[150] Y. Zhang, Z. Shi, W. Zuo, L. Yue, S. Liang, and X. Li, "Joint personalized markov chains with social network embedding for cold-start recommendation," *Neurocomputing*, vol. 386, pp. 208–220, 2020.

[151] Z. Kang, H. Xu, J. Hu, and X. Pei, "Learning dynamic graph embedding for traffic flow forecasting: A graph self-attentive method," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, IEEE, 2019, pp. 2570–2576.

[152] S. Zhou, "Empirical effect of graph embeddings on fraud detection/risk mitigation," *arXiv preprint arXiv:1903.05976*, 2019.

[153] S. J. Haggarty, K. M. Koeller, J. C. Wong, R. A. Butcher, and S. L. Schreiber, "Multidimensional chemical genetic analysis of diversity-oriented synthesis-derived deacetylase inhibitors using cell-based assays," *Chemistry & biology*, vol. 10, no. 5, pp. 383–396, 2003.

[154] M. J. Keiser, V. Setola, J. J. Irwin, C. Laggner, A. I. Abbas, S. J. Hufeisen, N. H. Jensen, M. B. Kuijer, R. C. Matos, T. B. Tran, *et al.*, "Predicting new molecular targets for known drugs," *Nature*, vol. 462, no. 7270, pp. 175–181, 2009.

[155] M. J. Keiser, B. L. Roth, B. N. Armbruster, P. Ernsberger, J. J. Irwin, and B. K. Shoichet, "Relating protein pharmacology by ligand chemistry," *Nature biotechnology*, vol. 25, no. 2, pp. 197–206, 2007.

[156] K. Bleakley and Y. Yamanishi, "Supervised prediction of drug–target interactions using bipartite local models," *Bioinformatics*, vol. 25, no. 18, pp. 2397–2403, 2009.

[157] Y. Liu, M. Wu, C. Miao, P. Zhao, and X.-L. Li, "Neighborhood regularized logistic matrix factorization for drug-target interaction prediction," *PLoS computational biology*, vol. 12, no. 2, e1004760, 2016.

[158] J.-P. Mei, C.-K. Kwoh, P. Yang, X.-L. Li, and J. Zheng, "Drug–target interaction prediction by learning from local information and neighbors," *Bioinformatics*, vol. 29, no. 2, pp. 238–245, 2013.

[159] F. Wan, L. Hong, A. Xiao, T. Jiang, and J. Zeng, "Neodti: Neural integration of neighbor information from a heterogeneous network for discovering new drug–target interactions," *Bioinformatics*, vol. 35, no. 1, pp. 104–111, 2019.

[160] L. Xie, L. Xie, S. L. Kinnings, and P. E. Bourne, "Novel computational approaches to polypharmacology as a means to define responses to individual drugs," *Annual review of pharmacology and toxicology*, vol. 52, pp. 361–379, 2012.

[161] Y. Yamanishi, M. Araki, A. Gutteridge, W. Honda, and M. Kanehisa, "Prediction of drug–target interaction networks from the integration of chemical and genomic spaces," *Bioinformatics*, vol. 24, no. 13, pp. i232–i240, 2008.

[162] N. Zong, H. Kim, V. Ngo, and O. Harismendy, "Deep mining heterogeneous networks of biomedical linked data to predict novel drug–target associations," *Bioinformatics*, vol. 33, no. 15, pp. 2337–2344, 2017.

[163] Y. Wang, P. Jiao, W. Wang, C. Lu, H. Liu, and B. Wang, "Bipartite network embedding via effective integration of explicit and implicit relations," in *International Conference on Database Systems for Advanced Applications*, 2019, pp. 435–451.

[164] L. Yu, C. Zhang, S. Pei, G. Sun, and X. Zhang, "Walkranker: A unified pairwise ranking model with multiple relations for item recommendation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[165] H. Deng, M. R. Lyu, and I. King, "A generalized co-hits algorithm and its application to bipartite graphs," in *Proceedings of the 15th ACM SIGKDD*, 2009, pp. 239–248.

[166] T. F. Smith, M. S. Waterman, *et al.*, "Identification of common molecular subsequences," *Journal of molecular biology*, vol. 147, no. 1, pp. 195–197, 1981.

[167] P. Willett, J. M. Barnard, and G. M. Downs, "Chemical similarity searching," *Journal of chemical information and computer sciences*, vol. 38, no. 6, pp. 983–996, 1998.

[168] M. Hattori, Y. Okuno, S. Goto, and M. Kanehisa, "Development of a chemical structure comparison method for integrated analysis of chemical and genomic information in the metabolic pathways," *Journal of the American Chemical Society*, vol. 125, no. 39, pp. 11 853–11 865, 2003.

[169] M. Vogt, D. Stumpfe, G. M. Maggiora, and J. Bajorath, "Lessons learned from the design of chemical space networks and opportunities for new applications," *Journal of computer-aided molecular design*, vol. 30, no. 3, pp. 191–208, 2016.

[170] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*, 1992, pp. 144–152.

[171] S. Archana and K. Elangovan, "Survey of classification techniques in data mining," *International Journal of Computer Science and Mobile Applications*, vol. 2, no. 2, pp. 65–71, 2014.

[172] D. S. Broomhead and D. Lowe, "Radial basis functions, multi-variable functional interpolation and adaptive networks," Royal Signals and Radar Establishment Malvern (United Kingdom), Tech. Rep., 1988.

[173] J. Platt *et al.*, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," *Advances in large margin classifiers*, vol. 10, no. 3, pp. 61–74, 1999.

[174] Z. Xia, L.-Y. Wu, X. Zhou, and S. T. Wong, "Semi-supervised drug-protein interaction prediction from heterogeneous biological spaces," in *BMC systems biology*, Springer, vol. 4, 2010, S6.

[175] C.-C. Chang and C.-J. Lin, "Libsvm: A library for support vector machines," *ACM transactions on intelligent systems and technology (TIST)*, vol. 2, no. 3, pp. 1–27, 2011.

[176] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE transactions on evolutionary computation*, vol. 1, no. 1, pp. 67–82, 1997.

[177] O. Martinho, R. Silva-Oliveira, V. Miranda-Goncalves, C. Clara, J. R. Almeida, A. L. Carvalho, J. T. Barata, and R. M. Reis, "In vitro and in vivo analysis of rtk inhibitor efficacy and identification of its novel targets in glioblastomas," *Translational oncology*, vol. 6, no. 2, 187–IN20, 2013.

[178] G. Noé, A. Bellesoeur, A. Thomas-Schoemann, S. Rangarajan, F. Naji, A. Puszkiel, O. Huillard, N. Saidu, L. Golmard, J. Alexandre, *et al.*, "Clinical and kinomic analysis identifies peripheral blood mononuclear cells as a potential pharmacodynamic biomarker in metastatic renal cell carcinoma patients treated with sunitinib," *Oncotarget*, vol. 7, no. 41, p. 67 507, 2016.

[179] R. Calero, E. Morchon, J. I. Johnsen, and R. Serrano, "Sunitinib suppress neuroblastoma growth through degradation of mycn and inhibition of angiogenesis," *PloS one*, vol. 9, no. 4, e95628, 2014.

[180] P. A. Harvey and L. A. Leinwand, "Oestrogen enhances cardiotoxicity induced by sunitinib by regulation of drug transport and metabolism," *Cardiovascular research*, vol. 107, no. 1, pp. 66–77, 2015.

[181] S.-A. Brown, L. Nhola, and J. Herrmann, "Cardiovascular toxicities of small molecule tyrosine kinase inhibitors: An opportunity for systems-based approaches," *Clinical Pharmacology & Therapeutics*, vol. 101, no. 1, pp. 65–80, 2017.

[182] B. Swathy and M. Banerjee, "Haloperidol induces pharmacoepigenetic response by modulating mirna expression, global dna methylation and expression profiles of methylation maintenance genes and genes involved in neurotransmission in neuronal cells," *PloS one*, vol. 12, no. 9, e0184209, 2017.

[183] Z. Liu, C. Huang, Y. Yu, P. Song, B. Fan, and J. Dong, "Dynamic representation learning for large-scale attributed networks," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 1005–1014.

[184] C. Hou, H. Zhang, K. Tang, and S. He, "Dynwalks: Global topology and recent changes awareness dynamic network embedding," *arXiv preprint arXiv:1907.11968*, 2019.

[185] B. Wang, A. Pourshafeie, M. Zitnik, J. Zhu, C. D. Bustamante, S. Batzoglou, and J. Leskovec, "Network enhancement as a general method to denoise weighted biological networks," *Nature communications*, vol. 9, no. 1, pp. 1–8, 2018.