LEARNING FROM ATTRIBUTED NETWORKS: EMBEDDING, THEORY, AND INTERACTIONS

A Thesis

by

XIAO HUANG

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2020

Major Subject: Computer Engineering

ABSTRACT

Networks are widely adopted to represent the relations between objects in many disciplines. In real-world scenarios, nodes are often associated with a rich set of data describing their characteristics, such as social networks with user-generated content. We model these systems as attributed networks. They are a unique data structure that simultaneously assesses networks and node individual attributes or content, and pervasive in practice. In this thesis, I present effective, scalable, and human-centric learning algorithms for attributed networks, to enable their actionable patterns to be easily accessible to data consumers.

Most machine learning algorithms make default assumptions that instances are independent of each other, and their features are in Euclidean space. These do not hold for attributed networks. To bridge the gap, attributed network embedding (ANE) aims to learn low-dimensional vectors to represent nodes, such that actionable patterns in original networks and node attributes can be preserved. The learned representations could be directly leveraged by off-the-shelf machine learning algorithms as feature vectors or hidden layers to conduct different tasks. I systematically developed a series of ANE algorithms, which could be categorized into four classes, including coupled spectral embedding, coupled-factorizations-based embedding, joint-random-walks-based embedding, and graph neural networks, to bridge the gap between large-scale networked data and off-the-shelf machine learning algorithms. On this basis, I developed interactive embedding to involve domain experts in advancing the ANE. Experts have a better cognition in the latent information such as domain knowledge and hidden relations. So we could learn from them and incorporate their knowledge into ANE.

My research enables data scientists and domain experts to effectively utilize the abundant but complex information. It broadly impacts fields such as Information Retrieval, Social Computing, Health Informatics, and Bioinformatics.

ACKNOWLEDGMENTS

It has been seven years since I left my country to fulfill my dream, a dream of becoming a professional researcher. Today, while towards the end of my Ph.D. thesis, I would like to express my sincere thanks and appreciation to all the people who have assisted me.

First, I would like to acknowledge my supervisor, Dr. Xia Hu, who made this thesis possible, for all his help, inspiration, and professional guidance. His unique thinking style and research methodology became a factor of motivation for my lifelong research career. I am so grateful.

Second, I would like to thank my thesis committee members, Dr. James Caverlee, Dr. Ricardo Gutierrez-Osuna, and Dr. Alexander Sprintson. Thanks for their time, advice, and support.

Also, I wish to express my deep and warm thanks to my labmates at the DATA (Data Analytics at Texas A&M) Lab, in the Department of Computer Science and Engineering, for accompanying and helping me all the time. Their valuable advice for my work has been extremely helpful.

Most importantly, I owe my sincerest gratitude to my family. Their love, understanding, and encouragement have provided me strong support for the completion of this thesis.

Finally, special thanks to the Texas A&M University and all the funding agencies, including National Science Foundation and Defense Advanced Research Projects Agency.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a dissertation committee consisting of Professor Xia Hu [advisor], Professor James Caverlee, and Professor Ricardo Gutierrez-Osuna, of the Department of Computer Science and Engineering and Professor Alexander Sprintson of the Department of Electrical and Computer Engineering.

All the work conducted for the dissertation was completed by the student independently.

Funding Sources

This work is, in part, supported by DARPA (#N66001-17-2-4031 and #W911NF-16-1-0565), and NSF (#IIS-1657196, #IIS-1718840, and #IIS-1750074). The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

NOMENCLATURE

$n = \|\mathcal{V}\| \in \mathbb{R}$	the number of nodes in the network
$m = \|\mathcal{U}\| \in \mathbb{R}$	the number of node attribute categories
$d \in \mathbb{R}$	the dimension of embedding representation vectors
$\mathcal{V}$	the set of nodes in the network
$\mathcal{E}$	the set of edges in the network
$\mathcal{U}$	the set of node attribute categories
$\mathbf{G} \in \mathbb{R}_{+}^{n \times n}$	the weighted adjacency matrix
$\mathbf{A} \in \mathbb{R}_{+}^{n \times m}$	the node attribute matrix
$\mathbf{S} \in \mathbb{R}^{n \times n}$	the affinity matrix defined by node attributes
$\mathbf{H} \in \mathbb{R}^{n \times d}$	the final embedding representation
$\mathbf{Z} \in \mathbb{R}^{n \times d}$	a copy of $\mathbf{H}$ for optimization
$N(i)$	the set of adjacent nodes of node i
$\mathrm{nnz}(\cdot)$	the number of nonzero elements in a matrix
$\mathcal{Q}_G$	node sequences sampled from the network $\mathbf{G}$
$\mathcal{Q}_A$	node sequences sampled from node attributes $\mathbf{A}$
$L \in \mathbb{R}_{+}$	the length of each truncated random walk
$\delta_j \in \mathcal{U}$	the j^{th} node attribute category
$\mathcal{A}$	the bipartite network constructed based on node attributes $\mathbf{A}$
$\mathcal{T}$	node index sequences sampled from $\mathbf{G}$ and $\mathcal{A}$
$\tau_i \in \mathcal{T}$	node sequences sampled for modeling node i

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

TABLE Page

1. INTRODUCTION[1]

1.1 Background and Motivation

With the prevalence of networked systems, such as social media [81], protein-protein interaction networks [26], and health care systems, network analysis has become an effective computational tool in practice. For example, in social media, service providers recommend potential friendships to users with link prediction techniques [5]. In proteomics, a surge of research is to identify the protein functions, and node classification [97] has been found to be significantly instructive. Traditionally, we could apply graph theory, such as shortest path and maximum flow, to perform the analysis. Examples include clustering based on normalized cuts and node classification based on label propagation. However, learning and prediction tasks in real-world networked systems have become incredibly complex, such as recommendations and advertising on large social networks with noisy user-generated content [5], completion of knowledge graphs with millions of entities [61], and fraud detection in financial transaction networks. Traditional graph theory can not directly satisfy the demands of real-world network analysis tasks.

To fill the gap and use off-the-shelf machine learning algorithms for addressing large and sophisticated networks, network embedding [91, 80] has been extensively investigated. Machine learning algorithms often make default assumptions that instances are independent of each other, and their features are in Euclidean space. These do not hold for networked data, because networks depict the node-to-node dependencies and are in non-Euclidean space. Network embedding targets at learning a low-dimensional vector to represent each node, such that actionable patterns in the original networks and side information can be preserved in the learned vectors. These low-

[1]Part of this chapter is reprinted with permission from "A General Embedding Framework for Heterogeneous Information Learning in Large-Scale Networks" by Xiao Huang, Jundong Li, Na Zou, and Xia Hu, 2018, ACM Transactions on Knowledge Discovery from Data, Vol. 1, No. 1, doi.org/10.1145/3241063, Copyright 2018 by Association for Computing Machinery, "Graph Recurrent Networks with Attributed Random Walks" by Xiao Huang, Qingquan Song, Yuening Li, and Xia Hu, 2019, Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Pages 732–740, Copyright 2019 by Association for Computing Machinery, and "Exploring Expert Cognition for Attributed Network Embedding" by Xiao Huang, Qingquan Song, Jundong Li, and Xia Hu, 2018, Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, Pages 270–278, Copyright 2018 by Association for Computing Machinery.

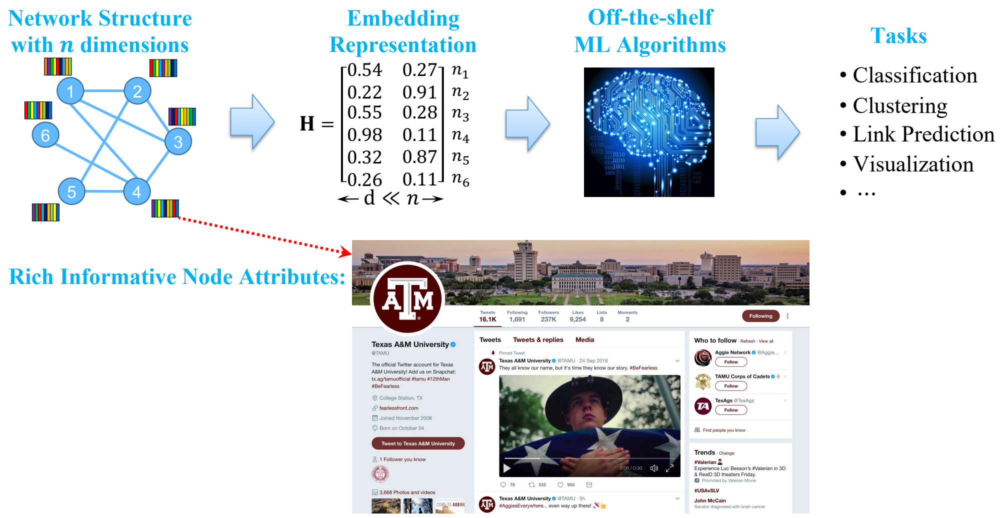

Network Structure with n dimensions
1
2
3
4
5
6
Embedding Representation
$\mathbf{H} = \begin{bmatrix} 0.54 & 0.27 \\ 0.22 & 0.91 \\ 0.55 & 0.28 \\ 0.98 & 0.11 \\ 0.32 & 0.87 \\ 0.26 & 0.11 \end{bmatrix} \begin{matrix} n_1 \\ n_2 \\ n_3 \\ n_4 \\ n_5 \\ n_6 \end{matrix}$
$\leftarrow d \ll n \rightarrow$
Off-the-shelf ML Algorithms
Tasks
• Classification
• Clustering
• Link Prediction
• Visualization
• …
Rich Informative Node Attributes:
Texas A&M University

Figure 1.1: In attributed networks, each node is associated with rich data reflecting its characteristics such as Texas A&M University. Attributed network embedding bridges the gap between real-world networked systems and off-the-shelf machine learning algorithms. [36]

dimensional representations could be directly leveraged by common machine learning algorithms as feature vectors or hidden layers to perform different tasks, such as node classification [89, 116], anomaly detection [63], community detection [105], and link prediction [21, 40].

While most of the existing network embedding algorithms focus on pure topological structure [24, 73], in real-world systems, nodes are often not pure vertices, but associated with plentiful attribute data, aka node attributes, describing their unique characteristics. Such systems are called attributed networks [35, 53]. For instance, with the popularity of social networking services, people not only make friends with each other to form online communities but also actively share opinions and post comments. Node attributes provide rich and highly related auxiliary information apart from network interactions for characterizing the node properties [53, 35, 86].

Node attributes are often informative and highly correlated with the network, and could be used to boost the network embedding. For instance, social science theories such as homophily hypothesis [64, 67] and social influence [98, 112] suggest that the network structure and node attributes tend to be mutually dependent on each other, i.e., the formation of one depends on and also influences the other. Figure 1.1 illustrates a toy attributed network of Twitter. The links capture the interactions among users, and network embedding aims to map them into continuous vector representations, which could be directly used by different applications. Meanwhile, a user like Texas A&M University also posts many tweets and comments that reflect her attributes. These posts have strong associations with her following relationships. Another example could be the Amazon product co-purchasing networks. Products purchased together tend to receive reviews with similar topics or written by the same group of customers [65]. Massive reviews also serve as a reliable resource for customers to find the properties of products and make informed purchase decisions. In addition, a number of data mining applications, such as sentiment analysis [33] and trust prediction [90], have been benefited by exploiting the correlations between geometrical structure and node attributes.

1.2 Attributed Network Embedding

Motivated by the observations, we formally propose attributed network embedding (ANE), as the bridges between networked systems and the off-the-shelf machine learning algorithms. Figure 1.1 illustrates its major goal. Given a set of nodes connected by a network and associated with node attributes, we aim to represent each node as a low-dimensional vector, such that the embedding representation could preserve the node proximity both in topological structure and node attributes. ANE is crucial to the field since its learned node representations could directly be employed as node features or hidden layers by off-the-shelf machine learning algorithms to perform various analysis tasks, such as node classification, network clustering, and anomaly detection.

However, it remains a challenging task to perform attributed network embedding. There are three basic challenges. First, the ever-growing data volume along with the complex data properties put demands on the scalability of algorithms. Real-world networks are often large-scale with a sheer amount of nodes and high-dimensional node attributes. For instance, there are over 68 million monthly active Twitter users in the United States as of 2019[2], and each user could post thousands of tweets. Some efforts [49, 81] have been devoted to leveraging the network structure and node attributes for seeking a joint low-rank latent representation. They either require eigen-decomposition [35, 48] with $\mathcal{O}(n^3)$ time complexity in each iteration (n denotes the total number

[2]https://www.statista.com/statistics/274564/monthly-active-twitter-users-in-the-united-states

of nodes) or employ gradient descent [81, 107, 116] which usually has a slow convergence rate. Second, assessing a vector representation for each node in the joint space of geometrical structure and a distinct type of information is difficult due to the bewildering combination of heterogeneous sources. A widely used approach to combining heterogeneous information sources is to calculate the node proximity matrix, i.e., similarities between nodes, based on each source, and conduct joint learning based on these homogeneous node proximities [35, 48]. But the computations of node proximity matrices would lead to high time and space complexity, not to mention the operations. Third, both of the topological structure and node attributes could be sparse, incomplete, and noisy, due to the imperfect data collection. It further exacerbates the joint embedding representation learning problem. Therefore, given the distinct characteristics of the data, existing methods cannot be directly applied to incorporate node attributes into the network embedding.

1.3 Human-in-the-loop Embedding

The helpfulness of expert cognition motivates us to investigate the potential of utilizing it to learn more informative embedding representations. We define the *expert cognition* as the intelligence-related information that experts know beyond the data, such as the understanding of domain knowledge [106], the awareness of conventions [14], and the perception of latent relations. An example would be the comprehension of bias in sampling, e.g., meteorologists know that tornadoes were more likely to be recorded in areas with more population [14]. Existing work such as explanation-based learning [16] and human-in-the-loop models [31, 44] suggest that the involvement of experts could increase the performance of many learning tasks. Expert cognition plays an essential role in advancing these data analysis. Since most existing network embedding algorithms are data-driven, we are motivated to explore how to take advantage of the expert cognition to enhance the performance of embedding. Sentiment analysis is a typical example of the utilization of expert cognition, which has been successfully applied to improve the performance of different recommendations [114]. In product review platforms such as Epinions and Slashdot, items are recommended to users based on their historical reviews. It has been shown that most reviews convey different levels of sentiment polarization [32], and the expert-created sentiment lexicons such as MPQA and SentiWordNet [87] are widely used to advance the review analysis and recommendation [18, 114].

Although some efforts have been devoted to directly learning expert cognition from the existing human-generated data [18], such concrete related data is still often limited and could be in different forms. Motivated by the success of active learning [84, 85] and interactive data mining [1, 31], in this paper, we propose to explore expert cognition via actively querying the experts. We aim to convert their abstract cognition into concrete answers, and incorporate them into attributed network embedding towards a more informative low-dimensional representation.

However, actively exploring expert cognition is a nontrivial task, with three major challenges as follows. First, expert cognition does not have a concrete format and is not easy to be measured, since it is related to the intelligence. Traditional solutions are to design specific algorithms according to the experts' understanding [16, 44]. It is hard to be generalized as they involve enormous engineering experimentations to transform the domain knowledge to algorithms. Second, in a real-world system, there are usually various types of expert cognition such as the comprehension of word meaning and the discernment of missing edges [31]. It is difficult to model all of them, or identify the types that can lead a significant performance gain in the embedding within a limited number of trials. It is also expensive to design specific models for different attributed networks and different embedding algorithms. A general way of modeling the expert cognition is essential. Third, the expert cognition is laborious to obtain as it involves humans in the whole process. Given a limited amount of human effort, we aim to design the queries in an effective way to maximize the total amount of learned expert cognition. Meanwhile, the returned answers from the experts could be heterogeneous with the attributed network. It is desired to have answers that could be easily incorporated into the embedding.

1.4 Thesis Major Contributions

This thesis addresses three primary scientific needs. (1) How to cope with the large scale issue while maintaining the effectiveness of modeling node proximity in the unified space composed of both network structure and node attributes? (2) While deep learning has been demonstrated

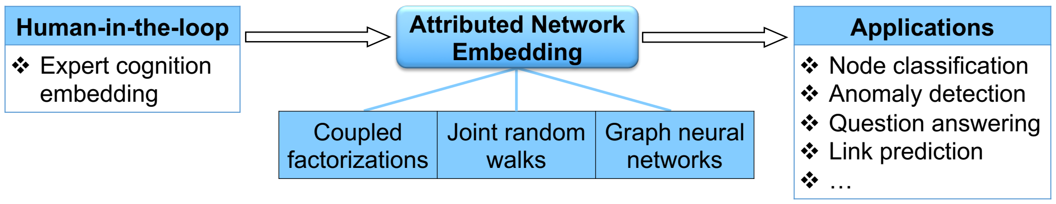


Figure 1.2: Summary of the contributions of this thesis.

to be effective in modeling various unstructured data such as images and audios, existing neural architectures could not be directly applied to embed the complex node interactions in attributed networks. How to perform effective and efficient deep ANE? (3) Cognition of domain experts could potentially be used to further boost the ANE. How to select the most meaningful queries to maximize the total amount of learned expert cognition, given a limited amount of human effort?

Figure 1.2 illustrates the major contributions of this thesis. We propose a series of attributed network embedding frameworks, to model the real-world networked systems within different scenarios. On the basis of ANE, we develop human-in-the-loop embedding to interactive with domain experts or data scientists and incorporate their cognition into the embedding. The learned representations would serve as infrastructure for developing various applications. Our key contributions can be summarized as follows.

- We propose an accelerated attributed network embedding framework - AANE, to tackle the aforementioned three challenges in attributed network embedding, i.e., large scale, heterogeneity, and sparsity. AANE accelerates the assessing of joint node proximity by decomposing the complex modeling and optimization into many independent simple sub-problems.

- Different real-world attributed networks have different characteristics, so algorithms with different theoretical foundations would have different performances. We develop another embedding framework - FeatWalk. It handles the heterogeneity via learning node similarities from node attributes. Since the computation of node similarities is time-consuming, we avoid it and propose an alternative way to simulate the similarity-based random walks among nodes to extract the local node proximity. This proposed joint walking mechanism is named AttriWalk, which considers node attributes as a bipartite network, and utilizes it to propel the random walks on the original topological structures more diverse and mitigate the tendency of converging to nodes with high centralities.

- We propose a neural-based embedding framework - GraphRNA. Based on the joint walking mechanism AttriWalk, we advance the graph convolutional networks to a more effective neural architecture, named graph recurrent neural networks. In this architecture, nodes are allowed to interact within the same way as they interact in the original attributed network.
- To help existing embedding frameworks learn abstract human knowledge, we present a general and concise framework - NEEC. It can provide experts a small number of carefully-designed concise queries, and their answers capture expert cognition systematically and could be directly added into the network to advance ANE. NEEC defines an effective algorithm to tackle the exploration and exploitation balancing in expert cognition learning.
- We empirically evaluate the effectiveness, efficiency, and generalizability of the proposed frameworks AANE, FeatWalk, GraphRNA, and NEEC, on several real-world datasets.

1.5 Related Work

This thesis is related to four research topics, including large-scale network embedding, attributed network embedding, network lasso, and pool-based active learning.

1.5.1 Large-scale network embedding

Large-scale network embedding has become an efficient tool to deal with real-world networks. The main question is how to efficiently learn low-dimensional representations for all vertices in a large network, such that the original geometrical information is recoverable. Efforts have been devoted from various aspects [24, 80, 91]. Tang and Liu [93] presented an edge-centric clustering scheme to facilitate the learning efficiency and alleviate the memory demand. Ahmed et al. [3] advanced a distributed matrix factorization algorithm to decompose large-scale graphs based on stochastic gradient descent. Tang et al. [91] improved the efficiency of stochastic gradient descent via an edge-sampling algorithm. The basic idea is to unfold a weighted edge into many binary edges by sampling each edge with a probability proportional to the weight. Ou et al. [73] designed a Jacobi-Davidson type algorithm to approximate and accelerate the singular value decomposition in the high-order proximity embedding. Wang et al. [103] involved a deep structure to embed both first and second order proximities of nodes. Grover and Leskovec [24] involved language modeling techniques to make the network embedding scalable. The basic idea is to conduct truncated random walks on a graph and analyze them as sentences. Two key parameters are defined to bias the walks towards broader or deeper space. Dong et al. [19] further advanced this idea to embed networks with heterogeneous types of nodes, and designed a novel heterogeneous skip-gram model to jointly learn structural and semantic correlations. Yang et al. [108] proposed to accelerate the embedding by approximating the network proximity via a theoretical bound. Most recently, a series of deep learning based embedding models [103, 99] have been explored. The focus of our paper is scalable models, and a survey on network embedding could be found in [13].

1.5.2 Attributed network embedding

research has been done to analyze attributed networks in various domains. It becomes increasingly promising to advance the learning performance by jointly exploiting geometrical structure and node attributes [33, 34, 90]. Tsur and Rappoport [98] improved the prediction of the spread of ideas by taking advantage of both content and topological features. Due to the complexity of attributed networks, nodes' properties and dependencies cannot be fully explained via these models. We roughly categorize existing attributed network embedding models into two classes.

Shallow attributed network embedding. To incorporate the topological structure and node attributes, attempts have been made from different aspects [35, 37]. Qi et al. [81] focused on multimedia objects and learned their latent semantic representations via jointly modeling their content information similarities and contextual links. Le and Lauw [49] advanced the topic modeling by incorporating the links between documents. Yang et al. [107] converted the network to pointwise mutual information and employed matrix tri-factorization to jointly embed the attributed network. Pan et al. [74] designed a coupled random walk based model with three objectives to incorporate the network, node attributes, and labels. Huang et al. [35] explored the potential of incorporating labels into the attributed network embedding. Liu et al. [63] performed local anomaly detection based on attributed network embedding, and accelerated the embedding process via a parallel mini-batch stochastic gradient descent. Li et al. [53] explored the attributed network embedding in a dynamic environment, and proposed an online framework based on the matrix perturbation theory. Yang et al. [109] proposed to leverage the Weisfeiler-Lehman graph kernels to learn binary vector representations of attributed networks. Several coupled matrix factorization based ANE methods [116, 56] have also been proposed.

Deep attributed network embedding. Given that rich training data becomes available, a series of effective deep ANE algorithms have been developed. Chang et al. [10] involved non-linear multilayered embedding functions to collaboratively embed the heterogeneous network and node content. Kipf and Welling [45, 46] introduced the graph convolutional networks, as a first-order approximation to the K-localized spectral graph convolution, which extended convolution operation from spatial domains to non-Euclidean spaces. Hamilton et al. [26] proposed the inductive representation learning, in which each node's representation is learned via aggregating all its neighbors' representations. Liang et al. [59] advanced the inductive representation learning by using a dual-input and dual-output neural architecture. Another line of research is to perform the joint embedding based on sophisticated objective functions [22, 51, 60, 115]. Recently, efforts have also been devoted to leveraging graph recurrent neural networks to conduct node or graph representation embedding [42, 88].

Attributed network analysis is different from multi-view learning [48]. The network structure is more than one angle of view. Its underlying properties are complex, including the connectivity, transitivity [73], first and higher order proximities [103], etc.

1.5.3 Network lasso

Network lasso was formally defined by Hallac et al. [25] as a simultaneous clustering and optimization problem. The key idea is to utilize the ℓ_2-norm distances of adjacent nodes as penalties and enforce nodes in the same cluster to have similar representations. The pioneer work of network lasso could be traced back to the fused lasso [95, 29] and convex clustering [78] problems.

Both of them can be viewed as special cases of network lasso and have been well explored. Lindsten et al. [62] demonstrated the equivalence between convex clustering and a convexification of k-means clustering. They utilized the off-the-shelf convex programming software CVX to handle the regularization. This solver is powerful in solving general convex problems but quite inefficient. Thus, Hocking et al. [28] introduced several efficient algorithms for convex clustering with three commonly used norms, ℓ_1, ℓ_2, and ℓ_∞. Recently, Chi and Lange [12] exploited the possibility of finding regularization paths of arbitrary norms by using ADMM and alternating minimization algorithm. These methods could only deal with thousands of nodes, and convex clustering often requires to include distances between every pair of nodes as penalties. We propose to use network lasso and symmetric matrix factorization jointly to perform the ANE. It is novel and non-trivial since the framework is required to be both separable and effective. Efforts also have been devoted to proving the monotonicity of the proposed updated rules.

1.5.4 Pool-based active learning

Pool-based active learning [84, 85], which learns from the oracle by inquiring labels. It aims to select a number of unlabeled instances from a pool, such that the classification performance could be maximized by using the labels of selected instances. The typical methods are based on uncertainty sampling [52, 96], query by committee [66], expected error and variance reduction [30], or expected model change [85]. Several efforts [7, 8] also have been devoted to balancing exploration and exploitation based on the multi-armed bandit algorithms.

Multi-armed bandit algorithms [2, 4] are widely used to perform online human behavior or knowledge learning. Radlinski et al. [82] exploited a bandit algorithm to conduct an online document ranking that can learn from users behavior. Pandey et al. [75] explored the scenario when selection strategies are dependent. Several contextual bandit based personalized recommendation algorithms [57, 104, 111] were developed to incorporate dynamic content and user information.

1.6 Thesis Organization and Notations

The goal of this thesis is to develop a series of attributed network embedding frameworks, to cover the scientific needs of real-world networked systems within different scenarios. In Chapter 2, we use the proposed scalable joint embedding framework named AANE, to demonstrate how to handle the three basic challenges in ANE, i.e., large scale, heterogeneity, and sparsity. AANE is motivated by the distributed matrix factorization and spectral embedding. In Chapter 3, we develop a joint-random-walks-based embedding framework named FeatWalk. In Chapter 4, we design a graph-neural-networks-based embedding framework named GraphRNA. In Chapter 5, on the basis of the proposed embedding frameworks, including AANE, FeatWalk, and GraphRNA, we develop a general way to learn from domain experts and leverage the learned knowledge to advance the embedding representations. In Chapter 6, we conclude the thesis and propose several advanced topics as the future work.

In this thesis, scalars are denoted by lowercase alphabets (e.g., n). Vectors are represented by boldface lowercase alphabets (e.g., $\mathbf{h}$). Matrices are denoted by boldface uppercase alphabets (e.g., $\mathbf{H}$). The i^{th} row of a matrix $\mathbf{H}$ is denoted by $\mathbf{h}_i$. The $(i,j)^{\text{th}}$ element of a matrix is denoted by h_{ij}. The transpose of $\mathbf{H}$ is represented as $\mathbf{H}^\top$. The dot product of two vectors is denoted by $\mathbf{a} \cdot \mathbf{b}$. The ℓ_2-norm of a vector is denoted by $\| \cdot \|_2$.The Frobenius norm of a matrix is represented as $\| \cdot \|_{\mathrm{F}}$. The identity matrix is denoted by $\mathbf{I}$. We use $\mathrm{nnz}(\cdot)$ to denote the number of nonzero elements. We use $\{\mathbf{r}_i\}$ to denote a sequence of vectors $\mathbf{r}_i$. The operation $\mathbf{z} = [\mathbf{x}, \mathbf{y}]$ denotes concatenating row vectors $\mathbf{x}$ and $\mathbf{y}$ into a new row vector $\mathbf{z}$.

2. COUPLED FACTORIZATION BASED ATTRIBUTED NETWORK EMBEDDING[1]

In this chapter, we first formally define the attributed network embedding problem, and then propose an accelerated joint embedding framework - AANE, to tackle the first proposed primary scientific need in Section 1.4.

2.1 Problem Statement

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{G})$ be a network, where $\mathcal{V}$ is a set of n nodes, and $\mathcal{E}$ is the set of edges. Each edge $(i, j) \in \mathcal{E}$ is associated with a positive weight $g_{ij} \in \mathbf{G}$. In this thesis, we focus on undirected networks. For directed ones, we follow the work of [55] to set g_{ij} and g_{ji} equal to their maximum value. A larger g_{ij} indicates higher similarity or stronger relationship between nodes i and j, and g_{ij} is defined as 0 if there is no edge. The set of adjacent nodes of node i is represented as $N(i)$. Let $\mathbf{A}$ be an $n \times m$ matrix that collects all node attributes, where m is the number of node attribute categories, and row $\mathbf{a}_i$ describes the attributes associated with node i.

Definition 2.1. (Node Attributes) In a network, we refer the extra textual, numerical, or image data that describes the characteristic features, records, or properties of each node as node attributes. Examples include posts and comments in social media, paper abstracts in academic networks, and property descriptions in protein-protein interaction networks.

Definition 2.2. (Node Proximity) It refers to the similarities between nodes defined by, (i) their edge weights in $\mathbf{G}$; (ii) their node attribute vectors, i.e., the rows of node attributes $\mathbf{A}$.

Based on the terminologies explained above, we formally define the problem of attributed network embedding as follows.

[1]This chapter is reprinted with permission from "A General Embedding Framework for Heterogeneous Information Learning in Large-Scale Networks" by Xiao Huang, Jundong Li, Na Zou, and Xia Hu, 2018, ACM Transactions on Knowledge Discovery from Data, Vol. 1, No. 1, doi.org/10.1145/3241063, Copyright 2018 by Association for Computing Machinery.

Dataset	Nodes (n)	Edges $(\|\mathcal{E}\|)$	Density	Attribute Categories (m)	Label (ℓ)
BlogCatalog	5,196	171,743	1.3e-002	8,189	6
Flickr	7,564	239,365	8.4e-003	12,047	9
Yelp	249,012	1,779,803	5.7e-005	20,000	11

Table 2.1: The detailed information of the three real-world attributed networks. [36]

Given a set of n nodes connected by a network $\mathcal{G}$ associated with edge weights $\mathbf{G}$ and node attributes $\mathbf{A}$, we aim to represent each node $i \in \mathcal{V}$ as a d-dimensional vector $\mathbf{h}_i$, such that this embedding representation $\mathbf{H}$ can preserve the node proximity both in topological structure $\mathcal{G}$ and node attributes $\mathbf{A}$. As a result, $\mathbf{H}$ could achieve better performance than $\mathbf{G}$ and $\mathbf{A}$ in terms of advancing other learning tasks.

2.2 Data Analysis on Real-world Networks

In this section, we verify the correlation between the network and node attributes on real-world attributed networks. Three datasets, i.e., BlogCatalog, Flickr, and Yelp are used in this work. All of them are publicly available and have been used in several previous work [35, 54]. The statistics of the three datasets are summarized in Table 2.1, with detailed descriptions as follows.

BlogCatalog is a blogger community, where users interact with each other and form a network. Users are allowed to generate keywords as a short description of their blogs. These keywords are served as node attributes. Users also register their blogs under predefined categories, and we set them as labels. The user with no follower or predefined category has been removed.

Flickr is an online community that people could share photos. These photographers could follow each other and form a network. We employ the tags specified on their images as the node attribute information. We set the groups that photographers joined as labels.

Yelp[2] is a social networking service, where crowd-sourced reviews about local businesses are shared. We employ users' friend relationships to form the network, and the bag-of-words model to represent users' reviews as node attributes. All local businesses are separated into eleven primary

[2] `https://www.yelp.com/dataset_challenge/dataset`

Dataset	Scenarios	CorrCoef	Intersect	p-value
BlogCatalog	Real-world	**3.69e-002**	**42**	**0.00e-016**
	RandomMean	3.14e-005	7.32	0.18
	RandomMax	1.40e-003	13	4.42e-016
Flickr	Real-world	**1.85e-002**	**25**	**0.00e-016**
	RandomMean	2.15e-005	3.56	0.49
	RandomMax	5.48e-004	9	3.37e-003

Table 2.2: Correlation between network and node attribute proximity w.r.t. the three metrics. [36]

categories, including Active Life, Arts & Entertainment, Food, Hotels & Travel, Nightlife, American Restaurants, Non-American Restaurants, Fast Food & Meat, Other Restaurants, Shopping, and Services. A user may have reviewed one or several businesses. We use the categories of these businesses as the user's labels.

The well-received studies such as homophily and social influences [67, 112] suggest that nodes with similar network structure would tend to have similar node attributes. We now verify this correlation experimentally. Since there is no formal definition for matrix correlation, we employ three types of metrics in the validation as follows.

Correlation Metrics: The first metric *CorrCoef* is defined as the Pearson correlation coefficient of all pairwise affinities in the network and node attribute space. There are $\binom{n}{2}$ node pairs in total, and we first calculate their cosine similarities based on network structure $\mathbf{W}$ and set them as the first group of variables $\mathbf{x} \in \mathbb{R}^{1\times\binom{n}{2}}$. Then we compute the node affinities w.r.t. node attributes $\mathbf{A}$ and set them as the second group $\mathbf{y} \in \mathbb{R}^{1\times\binom{n}{2}}$. We define the Pearson correlation coefficient of pairwise affinities $\mathbf{x}$ and $\mathbf{y}$ as CorrCoef. The second metric *Intersect* is defined as the number of common node pairs in the top 10^4 largest affinities in $\mathbf{x}$ and the top 10^4 largest affinities in $\mathbf{y}$. The third metric *p-value* is defined as the p-value of the null hypothesis as follows.

HYPOTHESIS 1. *There is no correlation between network affinities* $\mathbf{x}$ *and attribute affinities* $\mathbf{y}$.

To validate the correlation between the topological structure and node attributes, we include a randomly generated network as a baseline. This network has the same density as the real-world network, with undirected edges distributed randomly. We measure the correlation between the randomly generated network and real-world node attributes, and conduct 100 trials for each dataset. We define the average mean as *RandomMean* and the best performance as *RandomMax*. The results on BlogCatalog and Flickr are shown in Table 2.2. The result on Yelp is not available since it has too many pairwise affinities to be cached on a single machine. As we can see, on both datasets, CorrCoef and Intersect of the real-world networks are much larger than the ones of the randomly generated networks. The p-value of real-world network is 0.00e-016 on both datasets, which demonstrates that there is a significant relationship between the network proximity and node attribute proximity.

2.3 Accelerated Embedding Framework - AANE

We develop an effective and efficient framework AANE to deal with the challenges, i.e., large scale, heterogeneity, and sparsity. AANE jointly embeds the network structure and node attribute proximity in an effective way. Figure 2.1 illustrates its basic idea. Given a network with $n = 6$ nodes, it first decomposes attribute affinity $\mathbf{S}$ into the product of $\mathbf{H}$ and $\mathbf{H}^\top$. Meanwhile, it imposes an edge-based penalty into this decomposition such that connected nodes are close to each other in $\mathbf{H}$, and the closeness is controlled by the edge weights in $\mathbf{G}$. The goal is to make more similar nodes in the network space to be closer in $\mathbf{H}$. To accelerate the optimization, a distributed algorithm is proposed to separate the original problem into $2n = 12$ sub-problems with low complexity. The first $n = 6$ sub-problems are designed to be independent of each other, and the same as the last six. So all sub-problems can be assigned to $c = 3$ workers. In the final output, node 1 and node 3 are represented by similar vectors $[0.54, 0.27]$ and $[0.55, 0.28]$, which indicates that they are similar to each other in the original network and node attributes joint space.

2.3.1 Network topological structure modeling

To render the joint embedding representation $\mathbf{H}$ well-posed, AANE learns and preserves the node proximity in both the network and node attribute space.

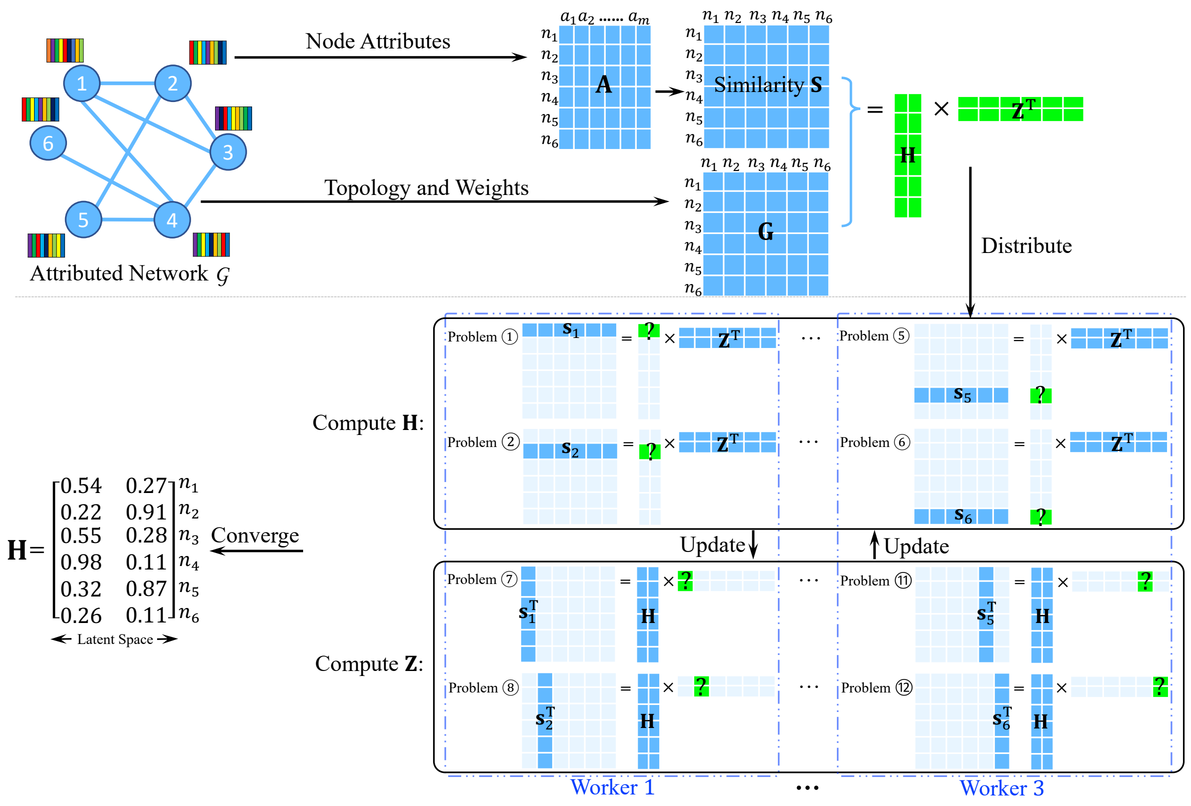


Figure 2.1: AANE represents nodes as low-dimensional vectors, based on the decomposition of attribute affinity matrix and the penalty of embedding difference between connected nodes. The optimization is split into $2n$ sub-problems with low complexity, which could be solved separately and parallelly by c workers. [36]

We first introduce how AANE models the network proximity via ℓ_2-norm to enable robust learning. The key idea is to drive nodes with more similar topological structure or connected by higher weights to have similar vector representations. It is based on two hypotheses [101, 28]. First, a graph-based mapping is assumed to be smooth across edges, especially for the regions of high density [20]. Second, the cluster hypothesis [70] suggests that similar nodes tend to cluster together. To achieve the goals, we propose the following loss function to minimize the embedding differences between all pairs of connected nodes,

$$\mathcal{J}_{\mathcal{G}} = \sum_{(i,j)\in\mathcal{E}} g_{ij}\|\mathbf{h}_i - \mathbf{h}_j\|_2, \tag{2.1}$$

where rows $\mathbf{h}_i$ and $\mathbf{h}_j$ are vector representations of node i and node j, and g_{ij} is the edge weight between the two. The key idea is that in order to minimize the penalty $g_{ij}\|\mathbf{h}_i - \mathbf{h}_j\|_2$, a larger weight g_{ij} is more likely to enforce the difference of $\mathbf{h}_i$ and $\mathbf{h}_j$ to be smaller. We employ ℓ_2-norm as the difference metric to alleviate the negative impacts resulted from outliers and missing data. The proposed network structure modeling enjoys several nice properties. First, it is in line with the fused lasso [95] and network lasso [25], which could induce sparsity in the differences of vector representations of similar nodes, and perform continuous edges selection similar to the continuous variable selection in lasso [94]. This is in compliance with the cluster hypothesis in graphs. Second, it can be generalized to model different types of networks, such as weighted and unweighted networks. Hence, it could be easily applied to many real-world applications.

2.3.2 Node attribute proximity modeling

As indicated by social science theories like homophily and social influences [64, 98], attribute information of nodes is tightly hinged with network topological structure. We now introduce how AANE makes $\mathbf{H}$ also well preserve the node proximity defined by the node attributes. Motivated by the symmetric matrix factorization [47], we propose to approximate attribute affinity matrix $\mathbf{S}$ with the product of $\mathbf{H}$ and $\mathbf{H}^\top$. The basic idea is to enforce the dot product of vector representations $\mathbf{h}_i$ and $\mathbf{h}_j$ to be the same as corresponding attribute similarity $\mathbf{s}_{ij}$. Mathematically, the loss function is defined as,

$$\mathcal{J}_A = \|\mathbf{S} - \mathbf{H}\mathbf{H}^\top\|_\mathrm{F}^2 = \sum_{i=1}^{n}\sum_{j=1}^{n}(\mathbf{s}_{ij} - \mathbf{h}_i\mathbf{h}_j^\top)^2, \tag{2.2}$$

where affinity matrix $\mathbf{S}$ could be calculated by a representative similarity measure. We use cosine similarity. The loss function $\mathcal{J}_A$ allows AANE to be optimized in a distributed manner.

2.3.3 Joint embedding representation learning

We have implemented two loss functions $\mathcal{J}_G$ and $\mathcal{J}_A$ to model the node proximity in network topological structure and node attributes. To make them complement each other towards a unified robust and informative space, we jointly model the two types of information in the following optimization problem,

$$\min_{\mathbf{H}} \ \mathcal{J} = \|\mathbf{S} - \mathbf{H}\mathbf{H}^\top\|_{\mathrm{F}}^2 + \lambda \sum_{(i,j)\in\mathcal{E}} g_{ij}\|\mathbf{h}_i - \mathbf{h}_j\|_2. \tag{2.3}$$

Scalar λ serves as an overall parameter that defines a trade-off between the contributions of network and attribute information. It is also a regularization parameter that balances number of clusters [25, 62]. An intuitive explanation is that when λ is close to 0, the network topology cannot affect the final result $\mathbf{H}$, so each node can be an isolated cluster. When λ is sufficiently large, the optimal solution will end up with same representations for all nodes, which forms a single cluster. This allows us to tune the number of clusters continuously. This number is not specified in network embedding, and tunability of λ is beneficial in this scenario.

2.3.4 A distributed optimization algorithm for acceleration

The proposed objective function not only jointly models network proximity and node attribute affinity, but also has a specially designed structure that enables it to be optimized in an efficient and distributed manner, i.e., $\mathcal{J}$ is separable for $\mathbf{h}_i$ and can be reformulated as a bi-convex optimization problem. Next, we propose an efficient algorithm to solve it.

First, we add a copy $\mathbf{Z} = \mathbf{H}$, then the first term in Eq. (2.3) can be rewritten as,

$$\|\mathbf{S} - \mathbf{H}\mathbf{Z}^\top\|_{\mathrm{F}}^2 = \sum_{i=1}^{n} \|\mathbf{s}_i - \mathbf{h}_i\mathbf{Z}^\top\|_2^2 = \sum_{i=1}^{n} \|\mathbf{s}_i^\top - \mathbf{H}\mathbf{z}_i^\top\|_2^2. \tag{2.4}$$

We further reformulate Eq. (2.3) into a linearly constrained problem as follows,

$$\begin{aligned} \min_{\mathbf{H}} \quad & \sum_{i=1}^{n} \|\mathbf{s}_i - \mathbf{h}_i\mathbf{Z}^\top\|_2^2 + \lambda \sum_{(i,j)\in\mathcal{E}} g_{ij}\|\mathbf{h}_i - \mathbf{z}_j\|_2, \\ \text{subject to} \quad & \mathbf{h}_i = \mathbf{z}_i,\ i = 1, \dots, n. \end{aligned} \tag{2.5}$$

This indicates that $\mathcal{J}$ is separable for both $\mathbf{h}_i$ and $\mathbf{z}_i$. Since ℓ_2-norm is convex, it is easy to verify that Eq. (2.5) is bi-convex, i.e., convex w.r.t. $\mathbf{h}_i$ when $\mathbf{Z}$ is fixed and convex w.r.t. $\mathbf{z}_i$ when $\mathbf{H}$ is fixed. However, it is infeasible to obtain closed-form solutions for these $2n$ sub-problems because of the linear constraint $\mathbf{H} = \mathbf{Z}$.

Second, we solve the above optimization by converting it into $2n$ simple update steps and one simple matrix update step, motivated by the distributed convex optimization technique - Alternating Direction Method of Multipliers (ADMM) [25, 9]. The augmented Lagrangian [27] of the objective function in Eq. (2.5) is formulated as follows,

$$\mathcal{L} = \sum_{i=1}^{n} \|\mathbf{s}_i - \mathbf{h}_i \mathbf{Z}^\top\|_2^2 + \lambda \sum_{(i,j)\in\mathcal{E}} g_{ij} \|\mathbf{h}_i - \mathbf{z}_j\|_2 + \frac{\rho}{2} \sum_{i=1}^{n} (\|\mathbf{h}_i - \mathbf{z}_i + \mathbf{u}_i\|_2^2 - \|\mathbf{u}_i\|_2^2), \tag{2.6}$$

where rows $\mathbf{u}_1, \ldots, \mathbf{u}_n \in \mathbb{R}^d$ are the scaled dual variables [9], and $\rho > 0$ is the penalty parameter. The minimizer of Eq. (2.5) is then converted to the saddle point of $\mathcal{L}$, which can be found by finding optimal $\mathbf{H}$, $\mathbf{Z}$, and $\mathbf{U}$.

Third, assume that, in iteration t, we have obtained the optimum of the three matrices as $\mathbf{H}^t$, $\mathbf{Z}^t$, and $\mathbf{U}^t$. Then, in iteration $t+1$, the calculation of $\mathbf{H}^{t+1}$ or $\mathbf{Z}^{t+1}$ could be separated into n independent and simple sub-problems as follows.

$$\begin{cases} \mathbf{h}_i^{t+1} = \underset{\mathbf{h}_i}{\operatorname{argmin}} \; (\|\mathbf{s}_i - \mathbf{h}_i {\mathbf{Z}^t}^\top\|_2^2 + \lambda \sum_{j\in N(i)} g_{ij} \|\mathbf{h}_i - \mathbf{z}_j^t\|_2 + 0.5\rho \|\mathbf{h}_i - \mathbf{z}_i^t + \mathbf{u}_i^t\|_2^2), & (2.7) \\ \mathbf{z}_i^{t+1} = \underset{\mathbf{z}_i}{\operatorname{argmin}} \; (\|\mathbf{s}_i - \mathbf{z}_i {\mathbf{H}^{t+1}}^\top\|_2^2 + \lambda \sum_{j\in N(i)} g_{ji} \|\mathbf{z}_i - \mathbf{h}_j^{t+1}\|_2 + 0.5\rho \|\mathbf{z}_i - \mathbf{h}_i^{t+1} - \mathbf{u}_i^t\|_2^2). & (2.8) \end{cases}$$

The update rule for $\mathbf{U}^{t+1}$ is defined as follows.

$$\mathbf{U}^{t+1} = \mathbf{U}^t + (\mathbf{H}^{t+1} - \mathbf{Z}^{t+1}). \tag{2.9}$$

We need to obtain all $\mathbf{h}_i^{t+1}$ before calculating $\mathbf{z}_i^{t+1}$, and update $\mathbf{U}^{t+1}$ after getting all the $\mathbf{h}_i^{t+1}$ and $\mathbf{z}_i^{t+1}$. However, the order of solving $\mathbf{h}_i^{t+1}$, for $i = 1, \ldots, n$, is not fixed, since they are independent of each other. When the machine capacity is limited, $\mathbf{s}_i$ could be calculated separately in each corresponding worker via equation $\mathbf{s}_i = \mathbf{a}_i \mathbf{A}^\top \oslash (q_i \mathbf{q})$, where notation $\oslash$ denotes the element-wise division. Vector $\mathbf{q}$ is the dot product of each node attribute vector and itself, i.e.,

$\mathbf{q} = [\sqrt{\mathbf{a}_1\mathbf{a}_1^\top}, \dots, \sqrt{\mathbf{a}_n\mathbf{a}_n^\top}]$.

2.3.4.1 Computation of $\mathbf{h}_i^{t+1}$ and $\mathbf{z}_i^{t+1}$

We introduce how to solve the problems in Eqs. (2.7) and (2.8). Although the function in Eq. (2.7) is convex, it is challenging to get the closed-form solution. We propose to approach the optimal $\mathbf{h}_i^{t+1}$ iteratively. There are several non-differentiable points at $\mathbf{h}_i = \mathbf{z}_j^t$ in Eq. (2.7), and the classical solution is to use subgradient methods. However, these methods usually converge slowly. Thus, we present an efficient heuristic approach to calculating $\mathbf{h}_i^{t+1}$.

We first define $\mathbf{h}_i^k = \mathbf{h}_i^t$ as the initial point, with $k = 0$. Then by taking the derivative of the function in Eq. (2.7) w.r.t. $\mathbf{h}_i$, and setting it to zero, we get an update rule for $\mathbf{h}_i^{k+1}$ as follows.

$$\mathbf{h}_i^{k+1} = (2\mathbf{s}_i\mathbf{Z}^t + \lambda \sum_{j\in\widetilde{N}(i)} \frac{g_{ij}\mathbf{z}_j^t}{\|\mathbf{h}_i^k - \mathbf{z}_j^t\|_2} + \rho\mathbf{z}_i^t - \rho\mathbf{u}_i^t) \times [2{\mathbf{Z}^t}^\top\mathbf{Z}^t + (\lambda \sum_{j\in\widetilde{N}(i)} \frac{g_{ij}}{\|\mathbf{h}_i^k - \mathbf{z}_j^t\|_2} + \rho)\mathbf{I}]^\dagger, \quad (2.10)$$

where $\widetilde{N}(i)$ is the set of adjacent nodes of node i with representations in $\mathbf{Z}^t$ not equal to $\mathbf{h}_i^k$, i.e., $\widetilde{N}(i) = \{j \in N(i) \text{ and } z_j^t \neq \mathbf{h}_i^k\}$. Here $\widetilde{N}(i)$ is defined to handle non-differentiable points. We employ $\|\mathbf{h}_i^k - \mathbf{z}_j^t\|_2$ to estimate the distance $\|\mathbf{h}_i^{k+1} - \mathbf{z}_j^t\|_2$, and the monotonically decreasing property of the update rule in Eq. (2.10) is proved in Theorem 2.3 and Corollary 2.4. It is proved that the objective function in Eq. (2.7) is optimal if and only if $\mathbf{h}_i^k = \mathbf{h}_i^{k+1}$ or $\mathbf{h}_i = \mathbf{z}_j^t$. Thus, we stop updating when they are close enough, and set the final one as $\mathbf{h}_i^{t+1}$. The proposed solution is heuristic because in a few extreme cases, $\mathbf{h}_i^t$ might equal to $\mathbf{z}_j^t$. We could get $\mathbf{z}_i^{t+1}$ in a similar way, with $\hat{N}(i) = \{j \in N(i) \text{ and } h_j^{t+1} \neq \mathbf{z}_i^k\}$, and the corresponding update rule of $\mathbf{z}_i^{k+1}$ is defined as,

$$\mathbf{z}_i^{k+1} = (2\mathbf{s}_i\mathbf{H}^{t+1} + \lambda \sum_{j\in\hat{N}(i)} \frac{g_{ij}\mathbf{h}_j^{t+1}}{\|\mathbf{z}_i^k - \mathbf{h}_j^{t+1}\|_2} + \rho\mathbf{h}_i^{t+1} + \rho\mathbf{u}_i^t) \times [2{\mathbf{H}^{t+1}}^\top\mathbf{H}^{t+1} + (\lambda \sum_{j\in\hat{N}(i)} \frac{g_{ij}}{\|\mathbf{z}_i^k - \mathbf{h}_j^{t+1}\|_2} + \rho)\mathbf{I}]^\dagger. \quad (2.11)$$

Theorem 2.3. (Monotonicity of Update Rules) *For any finite number of known vectors $\mathbf{z}_j \in \mathbb{R}^{1\times d}$ and a_j, and known $\mathbf{Q} \in \mathbb{R}^{d\times d}$ and $\mathbf{c} \in \mathbb{R}^{1\times d}$, given a function of $\mathbf{x} \in \mathbb{R}^{1\times d}$ defined as,*

$$f(\mathbf{x}) = \mathbf{x}\mathbf{Q}\mathbf{x}^\top + \mathbf{x}\mathbf{c}^\top + \textstyle\sum_j a_j\|\mathbf{x} - \mathbf{z}_j\|_2,$$

we have $f(\mathbf{x}^{k+1}) \leq f(\mathbf{x}^k)$ *for any pair of* $\{\mathbf{x}^k, \mathbf{x}^{k+1}\}$ *satisfying the rule as follows, with* $\mathbf{x}^k \neq \mathbf{z}_j$*,*

$$\mathbf{x}^{k+1} = (\textstyle\sum_j a_j b_j \mathbf{z}_j - \mathbf{c}) \times (2\mathbf{Q} + \textstyle\sum_j a_j b_j \mathbf{I})^{\dagger},$$

where $b_j = \frac{1}{\|\mathbf{x}^k - \mathbf{z}_j\|_2}$*. Furthermore,* $f(\mathbf{x})$ *is optimum if and only if* $\mathbf{x}^{k+1} = \mathbf{x}^k$ *or* $\mathbf{x} = \mathbf{z}_j$*.*

Corollary 2.4. *The update rule in Eq.* (2.10) *will monotonically decrease the objective function in Eq.* (2.7)*, and converge to the global optimum if and only if* $\mathbf{h}_i^{k+1} = \mathbf{h}_i^k$ *or* $\mathbf{h}_i = \mathbf{z}_j^t$*.*

2.3.4.2 Summary of the optimization

The distributed algorithm for optimizing the problem in AANE is described in Algorithm 1. To have an appropriate initialization, we set the initial embedding representation $\mathbf{H}^{t=0}$ as the left singular vectors of $\mathbf{G}^0$, where $\mathbf{G}^0$ is a matrix that randomly samples $2d$ columns of $\mathbf{G}$. We update $\mathbf{H}$ and $\mathbf{Z}$ iteratively until they are approximately equivalent and no longer change much in one iteration. The corresponding termination criterion is that primal residual $\mathbf{r}^t = \sum_{i=1}^n (\mathbf{h}_i^t - \mathbf{z}_i^t)$ and dual residual $\mathbf{s}^t = -\rho \sum_{i=1}^n (\mathbf{z}_i^t - \mathbf{z}_i^{t-1})$ should be sufficiently small [9]. To find $\mathbf{H}^{t+1}$, the optimization process is split into n number of sub-problems. These n update steps of $\mathbf{h}_i^{t+1}$ could be assigned to c workers in a distributed way as illustrated in Figure 2.1. We initialize $\mathbf{h}_i^k = \mathbf{h}_i^t$ and keep applying the update rule on $\mathbf{h}_i^{k+1}$ until it convergences, and the final result is set to be the new $\mathbf{h}_i^{t+1}$. Variable matrix $\mathbf{Z}^{t+1}$ is calculated in a similar way. $\mathbf{H}$ and $\mathbf{Z}$ are nonseparable in Eq. (2.5), and the convergence of nonseparable biconvex ADMM is still an open problem [23].

AANE enjoys several nice properties. First, it enables the n update steps for $\mathbf{h}_i^{t+1}$ (or $\mathbf{z}_i^{t+1}$), for $i = 1, \dots, n$, independent of each other. Thus, in each iteration, the global coordination could assign these tasks to available workers and collect the solutions from them without a fixed order. Second, all these small update steps have low complexity and converge fast. In a such way, we split the original complex embedding problem into $2n$ simple convex optimization sub-problems. Third, as it is typical for ADMM [9], Algorithm 1 tends to converge to a modest accuracy in a few iterations. The update of $\mathbf{h}_i^{k+1}$ (or $\mathbf{z}_i^{k+1}$) also converges rapidly, since the difference between

$\|\mathbf{h}_i^k - \mathbf{z}_j^t\|_2$ and $\|\mathbf{h}_i^{k+1} - \mathbf{z}_j^t\|_2$ is always small.

2.3.5 Complexity analysis

In the initialization, the time complexity for calculating singular vectors of an n by d matrix $\mathbf{G}_0$ is $\mathcal{O}(d^2 n)$. We denote the number of operations required to obtain the affinity matrix $\mathbf{S}$ as $\mathcal{T}_{\mathbf{S}}$. In each sub-problem, since we only need to compute ${\mathbf{Z}^t}^\top \mathbf{Z}^t$ once for all $\mathbf{h}_i^{t+1}$ per iteration, the update time for $\mathbf{h}_i$ should be $\mathcal{O}(d^3 + dn + d|N(i)|)$. Since $d \ll n$, this complexity could be

Algorithm 1: Accelerated Attributed Network Embedding - AANE [36]

```
   Input: $\mathbf{G}$, $\mathbf{A}$, $d$, $\epsilon$.
   Output: $d$-dimensional embedding representation $\mathbf{H}$.
1  Initialize $\mathbf{G}^0 \leftarrow$ First $2d$ columns of $\mathbf{G}$, $t = 0$, $\mathbf{H}^t \leftarrow$ Left singular vectors of $\mathbf{G}^0$;
2  Set $\mathbf{U}^t = \mathbf{0}$, $\mathbf{Z}^t = \mathbf{H}^t$, $\mathbf{q} = [\sqrt{\mathbf{a}_1\mathbf{a}_1^\top}, \ldots, \sqrt{\mathbf{a}_n\mathbf{a}_n^\top}]$, and $\mathbf{A} \leftarrow \mathbf{A} \oslash \mathbf{q}$;
3  repeat
4      Calculate ${\mathbf{Z}^t}^\top \mathbf{Z}^t$;
5      for $i = 1 : n$ do
           /* Assign $i^{\mathrm{th}}$ node to available worker, do:          */
6          Compute local node attribute affinity $\mathbf{s}_i = \mathbf{a}_i \mathbf{A}^\top$;
7          Set $k = 0$ and $\mathbf{h}_i^k = \mathbf{h}_i^t$;
8          repeat
9              Update $\mathbf{h}_i^{k+1}$ based on Eq. (2.10), and set $k = k + 1$;
10         until $\|\mathbf{h}_i^k - \mathbf{h}_i^{k-1}\|_2 \leq \epsilon$;
11         Set $\mathbf{h}_i^{t+1} = \mathbf{h}_i^k$;
12     end
13     Calculate ${\mathbf{H}^{t+1}}^\top \mathbf{H}^{t+1}$;
14     for $i = 1 : n$ do
           /* Assign $i^{\mathrm{th}}$ node to available worker, do:          */
15         Compute local node attribute affinity $\mathbf{s}_i = \mathbf{a}_i \mathbf{A}^\top$;
16         Set $k = 0$ and $\mathbf{z}_i^k = \mathbf{z}_i^t$;
17         repeat
18             Update $\mathbf{z}_i^{k+1}$ based on Eq. (2.11), and set $k = k + 1$;
19         until $\|\mathbf{z}_i^k - \mathbf{z}_i^{k-1}\|_2 \leq \epsilon$;
20         Set $\mathbf{z}_i^{t+1} = \mathbf{z}_i^k$;
21     end
22     Update $\mathbf{U}^{t+1} = \mathbf{U}^t + (\mathbf{H}^{t+1} - \mathbf{Z}^{t+1})$ and $t = t + 1$;
23 until $\|\mathbf{r}^t\|_2 \leq \epsilon^{pri}$ and $\|\mathbf{s}^t\|_2 \leq \epsilon^{dual}$;
24 return $\mathbf{H}^t$.
```

reduced to $\mathcal{O}(n)$. Therefore, the total time complexity of AANE is $\mathcal{O}(n+\mathcal{T}_{\mathbf{S}}+\frac{n^2}{c})$, which equals to $\mathcal{O}(n\text{nnz}(\mathbf{A})+\frac{n^2}{c})$. It should be noted that we avoid the computation of the similarity matrix of $\mathbf{G}$, which saves lots of time. Except for the space for storing the original $\mathbf{G}$ and $\mathbf{A}$ in the coordinator, it is easy to check that the space complexity of AANE is only $\mathcal{O}(n)$. Because the rule in Eq. (2.10) shows that, when calculating $\mathbf{H}$, only the low-dimensional matrix $\mathbf{Z}$ is replicated in each worker.

2.3.6 Attributed network embedding in streaming networks

The proposed framework AANE could also handle streaming networks. Given an optimal joint embedding representation $\mathbf{H}^*$, we could easily acquire the vector representations of newly included nodes. Assume we have a new node i with network features $\mathbf{w}_i$ and node attribute $\mathbf{a}_i$, we could obtain $\mathbf{h}_i$ readily via the objective function as follows,

$$\mathbf{h}_i = \underset{\mathbf{h}_i}{\text{argmin}} \; \|\mathbf{s}_i - \mathbf{h}_i\hat{\mathbf{H}}^\top\|_2^2 + 2\lambda \sum_{j\in N(i)} w_{ij}\|\mathbf{h}_i - \mathbf{h}_j^*\|_2, \tag{2.12}$$

where $\hat{\mathbf{H}}$ is a matrix that concatenates $\mathbf{H}^*$ and $\mathbf{h}_i$. Similarly, we could make a copy $\mathbf{z}_i = \mathbf{h}_i$ and approach the optimal solution of Eq. (2.12) via three sub-problems as follows.

$$\begin{cases} \mathbf{h}_i^{t+1} = \underset{\mathbf{h}_i}{\text{argmin}} \; (\|\mathbf{s}_i - \mathbf{h}_i\hat{\mathbf{Z}}^{t^\top}\|_2^2 + \lambda \sum_{j\in N(i)} w_{ij}\|\mathbf{h}_i - \mathbf{h}_j^*\|_2 + 0.5\rho\|\mathbf{h}_i - \mathbf{z}_i^t + \mathbf{u}_i^t\|_2^2), \\ \mathbf{z}_i^{t+1} = \underset{\mathbf{z}_i}{\text{argmin}} \; (\|1 - \mathbf{z}_i\mathbf{h}_i^{t+1^\top}\|_2^2 + \lambda \sum_{j\in N(i)} w_{ji}\|\mathbf{z}_i - \mathbf{h}_j^*\|_2 + 0.5\rho\|\mathbf{z}_i - \mathbf{h}_i^{t+1} - \mathbf{u}_i^t\|_2^2), \\ \mathbf{u}_i^{t+1} = \mathbf{u}_i^t + (\mathbf{h}_i^{t+1} - \mathbf{z}_i^{t+1}), \end{cases} \tag{2.13}$$

where $\hat{\mathbf{Z}}^t$ is a matrix that concatenates $\mathbf{H}^*$ and $\mathbf{z}_i^t$. Based on Theorem 2.3, it is straightforward to find the following update rules to iteratively solve the sub-problems in Eq. (2.13).

$$\begin{cases} \mathbf{h}_i^{k+1} = (2\mathbf{s}_i\hat{\mathbf{Z}}^t + \lambda\sum_{j\in N(i)} \frac{w_{ij}\mathbf{h}_j^*}{\|\mathbf{h}_i^k - \mathbf{h}_j^*\|_2} + \rho\mathbf{z}_i^t - \rho\mathbf{u}_i^t) \times [2\hat{\mathbf{Z}}^{t^\top}\hat{\mathbf{Z}}^t + (\lambda\sum_{j\in N(i)} \frac{w_{ij}}{\|\mathbf{h}_i^k - \mathbf{h}_j^*\|_2} + \rho)\mathbf{I}]^\dagger, \\ \mathbf{z}_i^{k+1} = (2\mathbf{h}_i^{t+1} + \lambda\sum_{j\in N(i)} \frac{w_{ij}\mathbf{h}_j^*}{\|\mathbf{z}_i^k - \mathbf{h}_j^*\|_2} + \rho\mathbf{h}_i^{t+1} + \rho\mathbf{u}_i^t) \times [2\mathbf{h}_i^{t+1^\top}\mathbf{h}_i^{t+1} + (\lambda\sum_{j\in N(i)} \frac{w_{ij}}{\|\mathbf{z}_i^k - \mathbf{h}_j^*\|_2} + \rho)\mathbf{I}]^\dagger. \end{cases} \tag{2.14}$$

2.4 Experiments

In this section, we empirically evaluate the effectiveness and efficiency of the proposed framework AANE. We aim at answering two questions as follows. (1) How effective is the embedding representation learned by AANE compared with other learning methods on real-world attributed networks? (2) How efficient is AANE compared with the state-of-the-art methods?

2.4.1 Baseline methods

AANE is compared with three categories of baseline methods. First, to evaluate the contribution of incorporating node attributes, two scalable network embedding methods are used for comparison, i.e., DeepWalk and LINE. Second, to study the contribution of incorporating the network, we include two methods for modeling pure node attributes, i.e., PCA and Spectral. Third, to investigate the effectiveness and efficiency of AANE, we compare it with two state-of-the-art attributed network learning methods, i.e., LCMF and MultiSpec. The detailed descriptions of these methods are listed as follows.

- *DeepWalk* [80]: It involves language modeling techniques to analyze the truncated random walks on a graph. It embeds the walking tracks as sentences and nodes as words.
- *LINE* [91]: It embeds the network into a latent space by sampling both one-hop and two-hop neighbors of each node. It is one of state-of-the-art scalable network embedding methods.
- *AANE_Net*: It treats the second-order network proximity as the node attributes, and employs AANE to incorporate it into the first-order proximity. Node attributes $\mathbf{A}$ are not used in it.
- *PCA* [43]: It is a classical dimensionality reduction technique. It takes the top d principal components of attribute matrix $\mathbf{A}$ as the learned representation.
- *Spectral* [101]: It embeds node attributes via two steps. First, it constructs a new graph with the cosine similarity of two nodes' attribute vectors as the corresponding edge weight. Second, it performs normalized spectral embedding on the constructed graph to learn $\mathbf{H}$.

- *LCMF* [116]: It learns a low-dimensional representation from linkage and content information by carrying out a joint matrix factorization on them.
- *MultiSpec* [48]: It treats network structure and node attributes as two views, and embeds them jointly by co-regularizing spectral clustering hypotheses across two views.
- *AANE_Stream*: Nodes in the test group arrive one by one. It employs the objective function in Eq. (2.12) to calculate the joint embedding representations of the streaming nodes.

2.4.2 Experimental setup

Following the widely adopted way of validating network embedding [80, 91], we evaluate AANE and baseline methods on the node classification task [89, 116]. The goal is to predict which category or categories a new node belongs to based on its low-dimensional representation and the learned classifier. We now introduce the experimental settings in detail.

We employ 5-fold cross-validation, i.e., randomly separate the entire nodes into a training group ($\mathbf{W}_{\text{train}}$, $\mathbf{A}_{\text{train}}$, $\mathbf{Y}_{\text{train}}$) and a test group ($\mathbf{W}_{\text{test}}$, $\mathbf{A}_{\text{test}}$, $\mathbf{Y}_{\text{test}}$), where $\mathbf{Y}$ denotes the labels. The edges between training group and test group are kept. To investigate the performance of a method, we apply it to both groups and learn vector representations $\mathbf{H}$ for all nodes, including $\mathbf{H}_{\text{train}}$ and $\mathbf{H}_{\text{test}}$. Since there are multiple label categories, we build a binary SVM classifier for each category based on $\mathbf{H}_{\text{train}}$ and $\mathbf{Y}_{\text{train}}$. At last, we perform the classification based on $\mathbf{H}_{\text{test}}$ and the learned SVM classifiers. The labels in $\mathbf{Y}_{\text{test}}$ serve as the ground truth.

The classification performance is measured via two commonly used evaluation criteria, micro-average and macro-average [41]. F-measure is a widely used metric for binary classification. Micro-average is the harmonic mean of average precision and average recall, i.e.,

$$\text{Micro-average} = \frac{\sum_{i=1}^{\ell} 2\text{TP}^{(i)}}{\sum_{i=1}^{\ell}(2\text{TP}^{(i)} + \text{FP}^{(i)} + \text{FN}^{(i)})}, \tag{2.15}$$

where $\text{TP}^{(i)}$, $\text{FP}^{(i)}$, and $\text{FN}^{(i)}$ denote the numbers of true positives, false positives, and false negatives in the i^{th} label category correspondingly. Macro-average is defined as an arithmetic average

		BlogCatalog				Flickr				Yelp-sub			
Training set percentage		10%	25%	50%	100%	10%	25%	50%	100%	10%	25%	50%	100%
nodes for embedding		1,455	2,079	3,118	5,196	2,118	3,026	4,538	7,564	13,945	19,921	29,881	49,802
Micro-average	DeepWalk	0.491	0.551	0.611	0.672	0.312	0.373	0.465	0.535	0.302	0.310	0.318	0.350
	LINE	0.433	0.545	0.624	0.684	0.259	0.332	0.421	0.516	0.230	0.243	0.264	0.294
	AANE_Net	0.556	0.628	0.690	0.747	0.315	0.397	0.496	0.626	0.369	0.373	0.274	0.247
	PCA	0.695	0.782	0.823	0.857	0.508	0.606	0.666	0.692	0.667	0.674	0.681	N.A.
	Spectral	0.717	0.791	0.841	0.869	0.698	0.771	0.813	0.846	0.670	0.683	N.A.	N.A.
	LCMF	0.778	0.849	0.888	0.902	0.576	0.676	0.725	0.749	0.668	0.680	0.686	N.A.
	MultiSpec	0.678	0.788	0.849	0.896	0.589	0.720	0.800	0.859	0.654	0.667	N.A.	N.A.
	AANE	**0.841**	**0.878**	**0.913**	**0.932**	**0.740**	**0.811**	**0.854**	**0.885**	**0.679**	**0.694**	**0.703**	**0.711**
	AANE_Stream	0.770	0.822	0.887	0.914	0.568	0.726	0.816	0.859	0.554	0.577	0.665	0.691
Macro-average	DeepWalk	0.489	0.548	0.606	0.665	0.310	0.371	0.462	0.530	0.139	0.159	0.215	0.275
	LINE	0.425	0.542	0.620	0.681	0.256	0.331	0.418	0.512	0.165	0.173	0.193	0.227
	AANE_Net	0.550	0.622	0.685	0.741	0.313	0.396	0.495	0.624	0.287	0.272	0.156	0.141
	PCA	0.691	0.780	0.821	0.855	0.510	0.612	0.671	0.696	0.591	0.599	0.605	N.A.
	Spectral	0.714	0.788	0.838	0.867	0.695	0.767	0.810	0.843	0.610	0.626	N.A.	N.A.
	LCMF	0.776	0.847	0.886	0.900	0.585	0.683	0.729	0.751	0.589	0.605	0.612	N.A.
	MultiSpec	0.677	0.787	0.847	0.895	0.589	0.722	0.802	0.859	0.578	0.589	N.A.	N.A.
	AANE	**0.836**	**0.875**	**0.912**	**0.930**	**0.743**	**0.814**	**0.852**	**0.883**	**0.630**	**0.645**	**0.656**	**0.663**
	AANE_Stream	0.770	0.815	0.884	0.912	0.567	0.727	0.815	0.858	0.503	0.538	0.613	0.646

Table 2.3: The classification performance of different methods on different datasets with $d = 100$. Training set percentage indicates the percentage of nodes in training group that are used for embedding. Nodes for embedding denotes the total number of nodes in the test group and nodes in the training group that are used. [36]

of F-measure of all ℓ label categories, i.e.,

$$\text{Macro-average} = \frac{1}{\ell}\sum_{i=1}^{\ell}\frac{2\text{TP}^{(i)}}{2\text{TP}^{(i)} + \text{FP}^{(i)} + \text{FN}^{(i)}}. \tag{2.16}$$

We follow suggestions of the original papers to set the parameters of baselines. If it is not specified, the embedding dimension d is set to be 100. All experimental results are the arithmetic average of ten test runs. We ran the experiments on a Dell OptiPlex 9030 i7-16GB desktop.

2.4.3 Effectiveness evaluation

To investigate the impacts of attribute information and how much it can improve the embedding representation, we compare AANE with all baseline methods. We also vary the nodes used for training as $\{10\%, 25\%, 50\%, 100\%\}$ of the entire training group to evaluate the effect brought by different sizes of training data. The classification performance w.r.t. training percentage is

Training Set Percentage		10%	25%	50%	100%
# nodes for embedding		69,723	99,605	149,407	249,012
Micro-average	DeepWalk	0.324	0.345	0.366	0.368
	LINE	0.295	0.313	0.336	0.354
	AANE	**0.698**	**0.709**	**0.711**	**0.714**
Macro-average	DeepWalk	0.239	0.254	0.266	0.260
	LINE	0.216	0.236	0.259	0.279
	AANE	**0.649**	**0.659**	**0.660**	**0.665**

Table 2.4: The classification performance of different scalable embedding methods on Yelp with $d = 100$, where baseline methods PCA, LCMF, and MultiSpec are infeasible. [36]

presented in Table 2.3. Since neither of existing attributed network learning methods is practicable on the Yelp dataset, we randomly select 20% of it and set as a new dataset, i.e., Yelp-sub, such that we are able to compare the effectiveness of different methods on it. Results of AANE on original Yelp dataset are shown in Table 2.4.

From the results, we have four major observations as follows. First, by taking advantage of node attributes, LCMF, MultiSpec, and AANE all achieve significantly better performance than network embedding methods on all datasets. For example, in BlogCatalog results, we can discover that incorporating attribute information allows AANE to achieve 38.7% gain over DeepWalk and 36.3% gain over LINE in terms of Micro-average score. Also, by performing joint embedding, AANE consistently outperforms PCA and Spectral on all datasets. For example, on BlogCatalog, AANE achieves 8.8% of improvements than PCA. Second, with the proposed formulation, AANE consistently outperforms LCMF and MultiSpec. For instance, on Flickr, AANE achieves 18.2% of improvements than LCMF, which can be explained by the fact that latent features learned by decomposing network matrix and attribute matrix are heterogeneous, and it is challenging to combine them. In addition, LCMF and MultiSpec are infeasible on the datasets Yelp. For instance, MultiSpec becomes impracticable as the number of nodes increases to 29,881, due to the high computational and memory requirement of eigen-decomposition. Third, when only network information is used, AANE_Net still outperforms DeepWalk and LINE on BlogCatalog and Flickr. For instance, on Flickr, AANE_Net achieves a gain of 17.0% over DeepWalk. AANE_Net does not

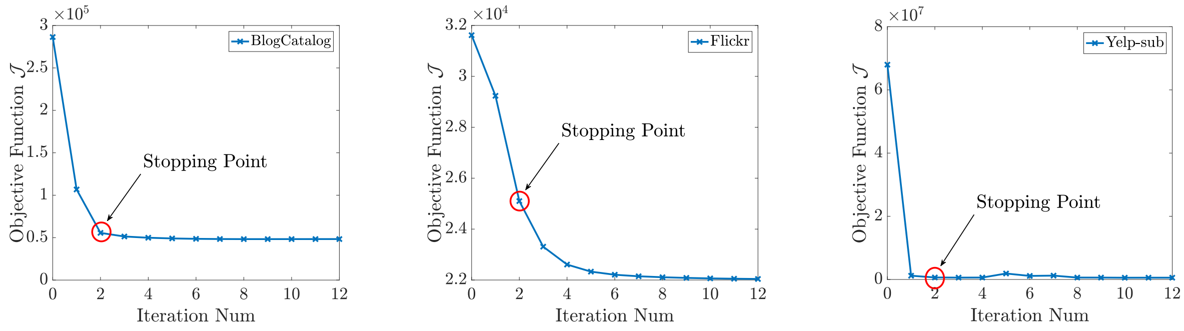


Figure 2.2: The augmented Lagrangian $\mathcal{L}$ in AANE w.r.t. the iteration number. [36]

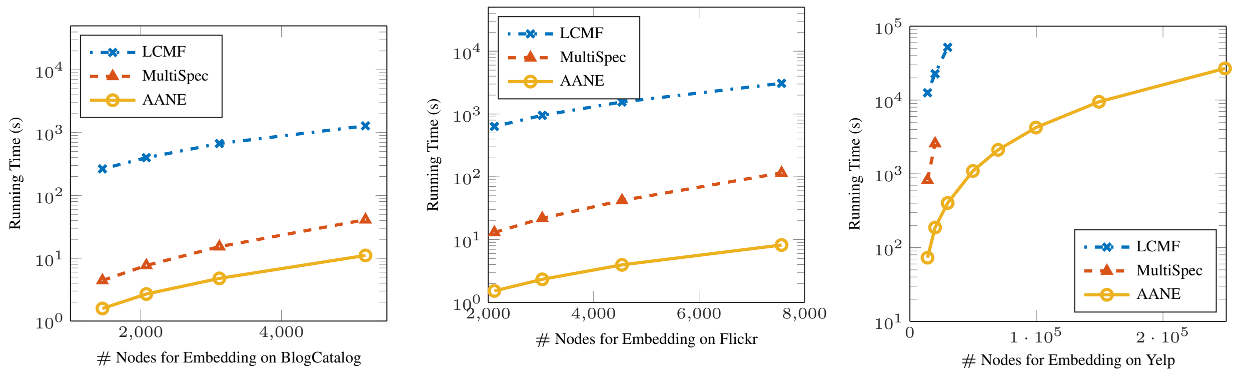


Figure 2.3: The running time of LCMF, MultiSpec and AANE when using a single thread. [36]

	BlogCatalog (sec)	Flickr (sec)	Yelp-sub (sec)
$c = 1$	26.301	33.751	1065.033
$c = 2$	14.233 (-45.9%)	17.510 (-48.1%)	581.544 (-45.4%)

Table 2.5: The running time of AANE w.r.t. the number of workers c on a dual-core processor. [36]

perform well on Yelp-sub since its network information is too noisy.

We further vary the percentage of the training group that is used for training. As shown in Tables 2.3 and 2.4, similar observations are made. LCMF, MultiSpec, and AANE outperform the two pure network embedding methods. AANE consistently achieves better performance than all baselines. We find that MultiSpec may perform worse than PCA. For instance, on Yelp-sub, PCA achieves a gain of 20.6% over MultiSpec when training percentage is 10%. One-tailed t-test results show that AANE is statistically significantly better (with p-value $\ll 0.01$) than all the baselines on all datasets. The p-value on BlogCatalog and Flickr are smaller than 9.5×10^{-14} and 1.6×10^{-12}.

2.4.4 Efficiency evaluation

To study the efficiency of the proposed framework AANE, we compare it with the two joint learning methods. Empirical results show that a near optimal solution of $\mathcal{J}$ is enough to guarantee $\mathbf{H}$ to be an informative embedding representation. Figure 2.2 shows the objective function $\mathcal{J}$ in AANE as a function of the iteration number on all the three datasets. As we can see, $\mathcal{J}$ decreases rapidly in the first two iterations. Therefore, in practice, only a few iterations are required for our framework AANE. If it is not specified, in the experiments, we stop the iteration at the stopping points shown in Figure 2.2. It should be noted that $\mathcal{J}$ might not decrease monotonically because of the non-differentiable points at $\mathbf{h}_i = \mathbf{z}_j^t$ in Eq. (2.7).

The computation time in logarithmic scale as a function of the number of input nodes on the three datasets are shown in Figure 2.3. The blue, red, and yellow dash-dot curves show the performance of LCMF, MultiSpec, and AANE respectively. From the results, we observe that AANE takes much less running time than LCMF and MultiSpec consistently. As the number of nodes increases, the performance difference raises up. The third subfigure shows that LCMF and MultiSpec have larger growth rates than AANE. They become infeasible when the number of nodes is greater than $49{,}802$ and $29{,}881$ respectively, due to the high computational cost and memory requirement of matrix factorization and eigen-decomposition. While the three methods are running within a single thread for a fair comparison, AANE could be implemented in multi-thread way as illustrated in Figure 2.1. This strategy could further improve the efficiency of AANE. We demonstrate the running time of AANE w.r.t. different numbers of workers c on a dual-core processor in Table 2.5. As we can see, on all the three datasets, the computation time of AANE is reduced by almost 50% when c is increased from 1 to 2. When $c = 1$, the running time on BlogCatalog and Flickr in Table 2.5 is larger than the one in Figure 2.3. It is because it takes extra time for the multi-thread version of AANE to set up the coordinator and workers. When n is large enough, e.g., on Yelp-sub, the running time with $c = 1$ in Table 2.5 is the same as the one in Figure 2.3. In summary, all these observations demonstrate the efficiency and scalability of AANE.

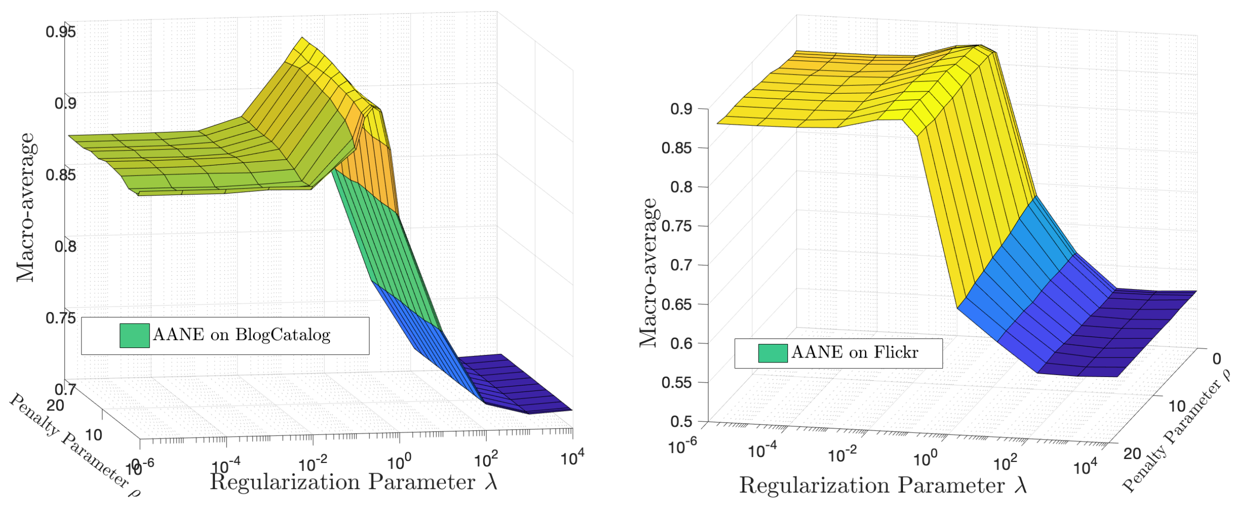


Figure 2.4: Impacts of regularization parameter λ and penalty parameter ρ on AANE. [36]

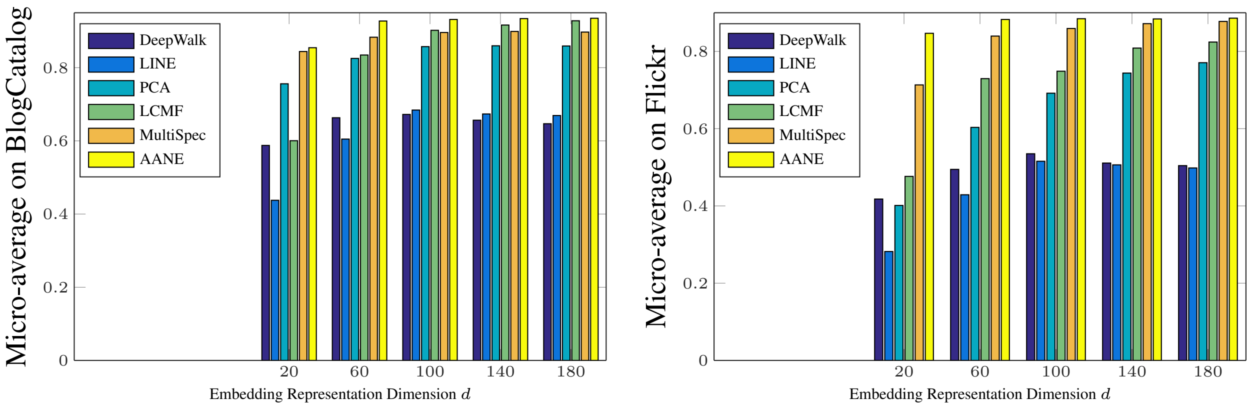


Figure 2.5: The classification performance of all methods on BlogCatalog and Flickr w.r.t. different dimension d. [36]

2.4.5 Parameter analysis

We study the impacts of important parameters, λ, ρ, and d. As discussed in Section 2.3.3, λ in AANE balances the contributions of network and node attributes. Penalty parameter ρ determines the amount of penalty from the linear constraint $\mathbf{H} = \mathbf{Z}$. To investigate the impacts of λ and ρ, we vary λ from 10^{-6} to 10^{4} and ρ from 0.1 to 20. Figure 2.4 shows the performance in terms of Macro-average as a function of λ and ρ on BlogCatalog and Flickr. We omit the results on Yelp since they are similar.

When $\lambda = 0$, network information has no impact as if all nodes are isolated. As λ increases, AANE starts to cluster nodes according to the topological structure, so the performance keeps improving. As shown in Figure 2.4, performance on BlogCatalog and Flickr achieves optimal when λ is close to 0.1. The performance decreases when λ is too large, since large λ tends to drive all nodes to have the same vector representation. The penalty parameter ρ has a limited impact on the performance of our framework AANE.

To study the impact of embedding dimension d, we vary it from 20 to 180. The classification performance of all methods in terms of Micro-average w.r.t. d is shown in Figure 2.5. We omit the results on Yelp since they are similar. From the results, we discover that observations made above hold undeviatingly as d varies. DeepWalk and LINE are always inferior to AANE and the two attributed network learning methods. AANE consistently outperforms all baseline methods. By increasing d, the performance of all methods first increases and then keeps stable. This shows that the low-dimensional representation performs well in capturing most of the meaningful information.

2.4.6 Effectiveness evaluation on streaming networks

AANE could also be applied to streaming networks. We make the nodes in the test group come one by one, and denote AANE on such streaming attributed network embedding as AANE_Stream. Its performance regards to different training set percentages is shown in Table 2.3. From the results, we observe that AANE_Stream achieves a comparable performance with the batch mode, i.e., AANE, on all the three datasets. For example, on Flickr, AANE_Stream performs only 2.8% worse than AANE when the training set percentage is 100%. The performance of AANE_Stream on Yelp-sub is worse than PCA since AANE relies on the topological structure to infer the vector representations of new nodes, while the network information of Yelp-sub is noisy.

3. JOINT RANDOM WALKS BASED ATTRIBUTED NETWORK EMBEDDING[1]

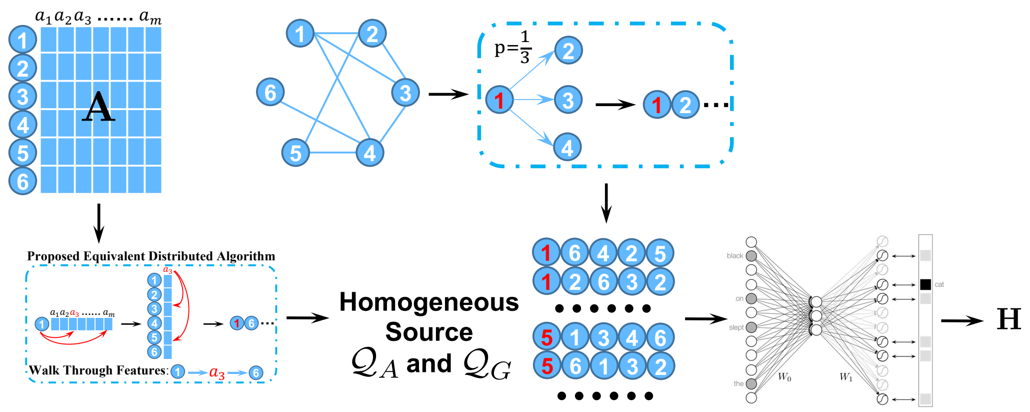
Random walks are widely adopted in network analysis. It converts geometric structures into structured sequences while alleviating the issue of sparsity. However, it is unclear how random walks could be developed for attributed networks towards an effective joint information extraction. Node attributes make the node interactions more complicated and are heterogeneous with respect to topological structures. In this chapter, we target at leveraging joint random walks to address the attributed network embedding problem. In such a way, embedding algorithms with different theoretical foundations would have different properties. Their performances vary in different real-world networks. We now introduce the second joint embedding framework - FeatWalk. The notations and definition of attributed network embedding are the same as Chapter 2.

3.1 Random Walks on All Features - FeatWalk

The network $\mathbf{G}$ and node attributes $\mathbf{A}$ are collected from different aspects. They are not only mutually dependent on and complement each other [86], but also heterogeneous with each other. To incorporate the heterogeneous information, a commonly used approach [113] is to calculate the node proximity based on each source respectively, and learn $\mathbf{H}$ from all node proximities jointly. It could effectively tackle the heterogeneity since node proximities are homogeneous. It should be noted that the proposed framework AANE in Chapter 2 also follows this effective approach.

To jointly embed the network $\mathbf{G}$ and node attributes $\mathbf{A}$, we propose an advanced attributed network embedding framework - *FeatWalk*. The node proximity defined by edge weights in $\mathbf{G}$ are homogeneous with the one defined by the similarity matrices of $\mathbf{A}$. FeatWalk achieves scalability by avoiding the computation of node similarities, and provides a distributed way to simulate similarity-based random walks among nodes. Figure 3.1 illustrates its main idea. FeatWalk first learns the node proximity defined by $\mathbf{G}$ via random walks and the one defined by $\mathbf{A}$ via AttriWalk.

[1]This chapter is reprinted with permission from "Large-Scale Heterogeneous Feature Embedding" by Xiao Huang, Qingquan Song, Fan Yang, and Xia Hu, 2019, Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33, No. 1, Copyright 2019 by Association for the Advancement of Artificial Intelligence.



$a_1 a_2 a_3 \ldots\ldots a_m$
A
p=$\frac{1}{3}$
Proposed Equivalent Distributed Algorithm
Walk Through Features:
Homogeneous Source
$\mathcal{Q}_A$ and $\mathcal{Q}_G$
H

Figure 3.1: FeatWalk projects the network and node attributes into homogeneous node sequences. [39]

Then it jointly incorporates them into a unified embedding representation $\mathbf{H}$. We conduct random walks on $\mathbf{G}$ and learn a set of node sequences $\mathcal{Q}_G$. As shown in Figure 3.2, to model the node proximity in $\mathbf{A}$, an intuitive solution is to construct a new graph $\mathbf{S}$ with node similarities as edge weights, and then perform random walks on $\mathbf{S}$ to learn a set of sequences $\mathcal{Q}_A$. However, $\mathbf{S}$ is often dense because of the common node attribute categories. As n keeps increasing, it would become too expensive to be manipulated. We propose a distributed algorithm - AttriWalk, which could obtain the same results as the intuitive solution but avoid the computation of $\mathbf{S}$. The learned $\mathcal{Q}_A$ consists of node indices that record the walking trajectories such as $[1, 6, 4, 2, 5]$. $\mathcal{Q}_A$ preserves the information in $\mathbf{A}$. Finally, $\mathcal{Q}_G$ and $\mathcal{Q}_A$ are homogeneous. By considering the node indices as words and sequences as sentences, a scalable word embedding technique is applied to $\mathcal{Q}_G$ and $\mathcal{Q}_A$ to learn a joint representation $\mathbf{H}$.

3.1.1 Network topological structure modeling

Given the network $\mathbf{G}$, we model its node proximity via conducting truncated random walk on it. We set each walk as the same length L, and collect all walking trajectories as a set of node sequences $\mathcal{Q}_G$. The probability of walking from node i to j is determined by the edge weight, i.e.,

$$P(i \rightarrow j) = \frac{g_{ij}}{\sum_{k=1}^{n} g_{ik}}. \tag{3.1}$$

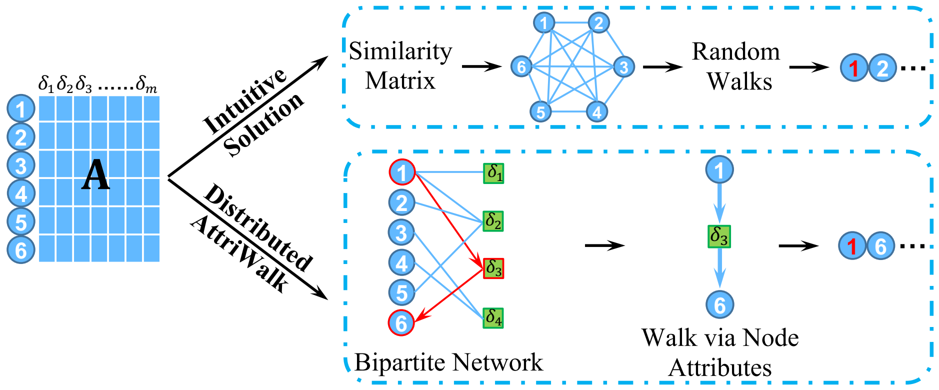


Figure 3.2: To avoid the computation of similarity matrices, FeatWalk performs equivalent random walks through node attributes, named AttriWalk. [39]

$\mathcal{Q}_G$ could capture the local node proximity, because as the number of learned sequences keeps increasing, the probability of index j follows index i in $\mathcal{Q}_G$ would approach to $P(i \to j)$. Thus, learning an embedding representation based on the indices' co-occurrence probabilities is equivalent to the one based on the nodes' linking probabilities.

We set the total number of walks assigned to model $\mathbf{G}$ as W_G. To make sure local proximities of all nodes could be sampled, we select each node as the initial index in turns. However, the number of random walks using node i as an initial index, i.e., b_i, is not fixed. We assign it based on the complexity of corresponding node, which is defined as follows,

$$b_i = \frac{\text{nnz}(\mathbf{g}_i) W_G}{\sum_{p=1}^{n} \text{nnz}(\mathbf{g}_p)}, \tag{3.2}$$

We design the function in (3.2) based on two assumptions. First, nodes with more edges would require more random walks to simulate its linking probabilities. For example, if in $\mathbf{G}$, nodes 4 and 6 have four and one edges respectively. We need to walk from node 4 at least four times and from node 6 at least one time to capture their relationships with all neighbors. Second, in a network, nodes with more edges tend to be more important [70]. To make the local proximity of important nodes well-preserved, we sample more sequences for them. Thus, we assign the numbers of walks $\{b_i\}$ based on the complexity.

3.1.2 Node attribute proximity modeling

As shown in Figure 3.2, to model the node proximity defined by $\mathbf{A}$, an intuitive solution is to compute its similarity matrix to construct a new graph S, and perform random walks on S. The weight of edge between nodes i and j in S is defined as the similarity between $\mathbf{a}_i$ and $\mathbf{a}_j$. However, this intuitive solution has several problems. First, as n increases, the size of S would increase exponentially. The calculation, storage, and manipulations of S would become expensive. Second, S is often quite dense, which makes the random walks on S inefficient. For example, when creating node attributes for Twitter users based on their tweets, commonly used words such as "good" and "think" would make S close to a clique. Then the time complexity of sampling a neighbor from s_i would become $\mathcal{O}(n)$ [17], which is expensive. Therefore, we propose a distributed algorithm - AttriWalk, which solves these problems by avoiding the computation of S.

3.1.2.1 Random walks for node attributes - AttriWalk

Since the computation of and operations on S are expensive, we design an alternative way to simulate the similarity-based random walks on S, with details as follows.

I. We normalize the node attribute matrix $\mathbf{A}$. We use ℓ_2 norm to normalize each row of $\mathbf{A}$ and get $\tilde{\mathbf{A}}$. Since it is hard for random walks to simulate the probabilities with small values, we remove the small elements in $\tilde{\mathbf{A}}$, i.e., elements smaller than $\beta\text{Mean}(\tilde{\mathbf{A}})$, and get $\hat{\mathbf{A}}$. $\text{Mean}(\tilde{\mathbf{A}})$ denotes the mean value of all elements in $\tilde{\mathbf{A}}$ and β is a threshold value. We use ℓ_1 norm to normalize each row of $\hat{\mathbf{A}}$ and get a new matrix $\bar{\mathbf{A}}$, i.e.,

$$\bar{a}_{ik} = \frac{\hat{a}_{ik}}{\sum_{p=1}^{m} \hat{a}_{ip}}, \tag{3.3}$$

where m is the total number of node attribute categories in $\mathbf{A}$.

II. We collect all the m node attribute categories as a set, denoted as $\mathcal{U}$. By considering each node attribute category $\delta_k \in \mathcal{U}$ as a node, we could define a new bipartite network as $\mathcal{A} \triangleq (\mathcal{V}, \mathcal{U}, \mathcal{E}_{\mathcal{A}})$, where $\mathcal{E}_{\mathcal{A}}$ represents the corresponding edge set. There exists an edge between nodes $i \in \mathcal{V}$ and $\delta_k \in \mathcal{U}$, if node i contains attributes δ_k. The weight of this edge is defined as $\bar{a}_{ik}$. Assume that currently we have jumped to a node $i \in \mathcal{V}$. As illustrated in Figure 3.2, we jump to a node $\delta_k \in \mathcal{U}$ with a probability,

$$P(i \to \delta_k) = \frac{\bar{a}_{ik}}{\sum_{p=1}^{m} \bar{a}_{ip}} = \bar{a}_{ik}. \tag{3.4}$$

III. Given that δ_k is selected, we jump to a node $j \in \mathcal{V}$, with a probability defined as,

$$P(\delta_k \to j) = \frac{\bar{a}_{jk}}{\sum_{q=1}^{n} \bar{a}_{qk}}. \tag{3.5}$$

In such a way, we accomplish the walk from node i to j. Similar to the random walks on $\mathbf{G}$, the length of each walk is set as L. The number of walks using i as the initial index is defined as,

$$\hat{b}_i = \frac{\text{nnz}(\bar{\mathbf{a}}_i) W_A}{\sum_{p=1}^{n} \text{nnz}(\bar{\mathbf{a}}_p)}. \tag{3.6}$$

3.1.2.2 Theoretical analysis of AttriWalk

The output of AttriWalk is equivalent to the output of random walks on a special node proximity $\bar{\mathbf{S}}$. The proof of Theorem 3.1 is in Appendix A.

Theorem 3.1. *In AttriWalk, the probability of walking from node $i \in \mathcal{V}$ to another node $j \in \mathcal{V}$ through any attribute categories, follow a similarity matrix $\bar{\mathbf{S}}$ defined as follows.*

$$\bar{\mathbf{S}} = \bar{\mathbf{A}}\mathbf{D}\bar{\mathbf{A}}^\top, \tag{3.7}$$

where $\mathbf{D}$ is a diagonal matrix, with $d_{kk} = \frac{1}{\sum_{q=1}^{n} \bar{a}_{qk}}$.

3.1.3 Joint embedding with all random walks

All sequences in $\mathcal{Q}_G$ and $\mathcal{Q}_A$ are homogeneous with each other. We put them together to jointly perform the node proximity learning. Let W be the total number of sequences that we could sample. To balance the contributions of $\mathbf{G}$ and $\mathbf{A}$, W_G and W_A are defined as,

$$W_G = \alpha W \quad \text{and} \quad W_A = (1-\alpha) W. \tag{3.8}$$

By applying a scalable word embedding method [69], we could learn a joint embedding representation $\mathbf{H}$ from $\mathcal{Q}_G$ and $\mathcal{Q}_A$. Word embedding [79] aims to map each word in a set of sentences into a low-dimensional vector, so that words with similar semantic meaning would have similar vector representations. Since massive amounts of documents are available in practice, many scalable word embedding algorithms such as word2vec [69] and GloVe [79] have been proposed. We could take advantage of these efficient algorithms to learn joint low-dimensional representations of nodes from $\mathcal{Q}_G$ and $\mathcal{Q}_A$, by considering the nodes as words in sentences.

It is straightforward to extend FeatWalk to multiple networks and multiple types of node attributes. We could perform random walks on each network and AttriWalk on each type of node attributes, to learn multiple sets of homogeneous node sequences. Then a joint $\mathbf{H}$ could be learned from these sets of node sequences.

3.1.4 Complexity analysis

Let $\mathcal{T}$ denote the number of operations required to obtain $\mathbf{H}$ based on the W learned sentences. In the tasks of sampling from a discrete probability distribution, we use the alias method [17]. The time complexity of the setup of alias method is $\mathcal{O}(\text{nnz}(\mathbf{G}))$ or $\mathcal{O}(\text{nnz}(\mathbf{A}))$ and each sampling takes $\mathcal{O}(1)$ time. Then, the time complexity of modeling $\mathbf{G}$ and $\mathbf{A}$ is $\mathcal{O}(\text{nnz}(\mathbf{G})+\text{nnz}(\mathbf{A})+WL)$. Thus, the time complexity of generating all sequences is linear with the numbers of nonzero entries in $\mathbf{G}$ and $\mathbf{A}$. The total time complexity of FeatWalk is $\mathcal{O}(\text{nnz}(\mathbf{G}) + \text{nnz}(\mathbf{A}) + WL + \mathcal{T})$. It should be noted that the processes of learning any two sequences are independent of each other. We could implement them in parallel to further accelerate FeatWalk.

3.2 Experiments

We now empirically validate the efficiency and effectiveness of FeatWalk. There are three major questions we aim to answer. (1) How efficient is FeatWalk in performing heterogeneous information embedding compared with the state-of-the-art methods? (2) How effective is the representations learned by FeatWalk compared with other embedding methods in applications such as classification? (3) What are the impacts of parameters α, the window size, L, the number of sentences per node W/n, and d on FeatWalk?

3.2.1 Three real-world datasets

The three real-world datasets that we employed in the experiments are all publicly available, including Flickr, ACM, and Yelp. We have introduced Flickr and Yelp in Chapter 2.

ACM [92]: $48{,}579$ papers published in ACM before 2016 are utilized as nodes. We construct an undirected $\mathbf{G}$ based on the citation links, with $288{,}374$ edges in total. We apply the bag-of-words model to represent the abstracts as node attributes, with $m = 10{,}000$. All papers are from nine areas as follows. Artificial Intelligence (AAAI, IJCAI, etc.), Computer Vision (TPAMI, CVPR, etc.), Computational Linguistics (ACL, EMNLP, etc.), Data Mining (KDD, WSDM, etc.), Databases (VLDB, TKDE, etc.), Human Computer Interaction (CHI, UIST, etc.), Information Retrieval (CIKM, SIGIR, etc.), Machine Learning (ICML, COLT, etc.), Robotics (ICRA, IROS, etc.). We employ areas as labels.

3.2.2 Baseline methods

To study the performance of FeatWalk, we compare it with three categories of baselines. First, to investigate the impact of node attributes, we include three single feature embedding methods, i.e., NMF, Spectral, FeatWalk_A. Second, to study the impact of the networks, we include two network embedding methods, i.e., DeepWalk and LINE. Third, to analyze the efficiency and effectiveness of FeatWalk, we include three state-of-the-art heterogeneous feature embedding methods, i.e., LCMF, MultiSpec, and AANE, and a variation of FeatWalk named w/o_FW. Several baselines have been introduced in Chapter 2, including DeepWalk, LINE, MultiSpec, and AANE.

- *NMF* [77]: It is a scalable version of non-negative matrix factorization, optimized by the hierarchical alternating least squares algorithms. It reduces the dimension of $\mathbf{A}$ to learn $\mathbf{H}$.

- *Spectral* [101]: It calculates the cosine similarities between vectors $\mathbf{a}_i$ to construct a new graph, and applies spectral embedding on it to learn $\mathbf{H}$.

- *FeatWalk_A*: It learns the representation $\mathbf{H}$ only from node attributes $\mathbf{A}$ via FeatWalk.

- *w/o_FW*: FeatWalk without using AttriWalk. Instead, it calculates $\mathbf{S}$ directly, and performs random walks on $\mathbf{S}$.

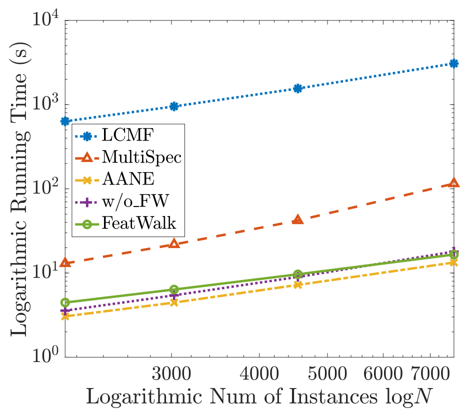
3.2.3 Experimental settings

We use the same experimental settings as Chapter 2. We use the original papers' default settings to determine the parameters of baselines. FeatWalk_A and w/o_FW use the same parameters as FeatWalk. If it is not specified, d is set as 100 and 100% of the nodes in the training group are used. We performed ten test runs and used the arithmetic average as the final experimental results.

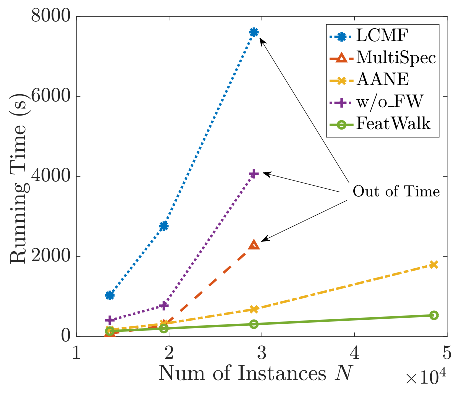
3.2.4 Efficiency of FeatWalk

To investigate the first question proposed at the beginning of this section, we compare FeatWalk with the three state-of-the-art heterogeneous feature embedding methods and w/o_FW. The running time of all methods as a function of the number of nodes n on Flickr, ACM, and Yelp is shown in Figure 3.3.

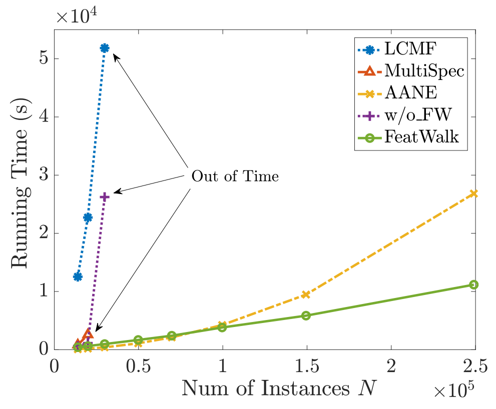
From the results in Figure 3.3, we have three major observations. First, the running time of FeatWalk is almost linear to n, which demonstrates its scalability. For example, in Figure 3.3c, the slope of the green curve (FeatWalk) remains invariable as n increases. As n keeps increasing, LCMF, MultiSpec, and w/o_FW run out of time since their running time increase exponentially. Second, the distributed sampling algorithm AttriWalk has significantly accelerated FeatWalk and made it scalable. When n is small, as shown in Figure 3.3a, w/o_FW has almost the same running time as FeatWalk, since the manipulations of similarity matrix $\mathbf{S}$ are cheap at this time. However, when n keeps increasing, as shown in Figure 3.3b, the running time of w/o_FW increases exponentially until it runs out of time. FeatWalk has significantly less running time than w/o_FW on both ACM and Yelp. Third, FeatWalk always has the least running time when n is large. When n is small, AANE might have less running time than FeatWalk. FeatWalk has almost the same running time as AANE on Flickr, and is always faster than AANE on ACM. On Yelp, FeatWalk needs more running time than AANE when $N <$ 82,000, but it becomes faster than AANE when $N \geq$ 82,000 since it is almost linear to n.



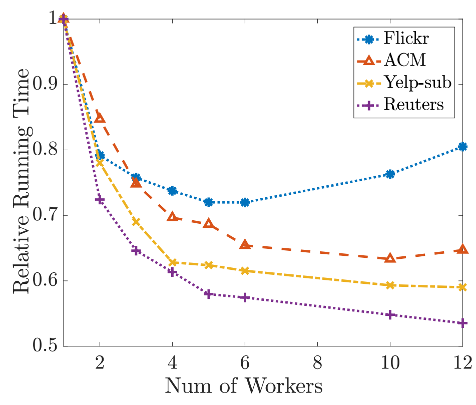
(a) Flickr



(b) ACM



(c) Yelp



(d) Number of workers

Figure 3.3: The running time of FeatWalk and different joint embedding methods. [39]

The running time of FeatWalk can be further reduced by using the multi-thread implementation. Figure 3.3d shows the relative running time of FeatWalk as a function of the number of workers c on all datasets. From the results, we have three observations. First, FeatWalk becomes more efficient as c increases. Second, as c increases from 1 to 2, the running time decreases less than 50%. It is because the multi-thread implementation only accelerates the learning of sequences, not the word embedding process. Third, the running time of FeatWalk on Flickr keeps increasing when $c \leq 5$. It is because $n = 7{,}564$ is relatively small and it takes extra time for the communications between the coordinator and workers. It should be noted that word2vec can be replaced by other more efficient algorithms to make FeatWalk faster.

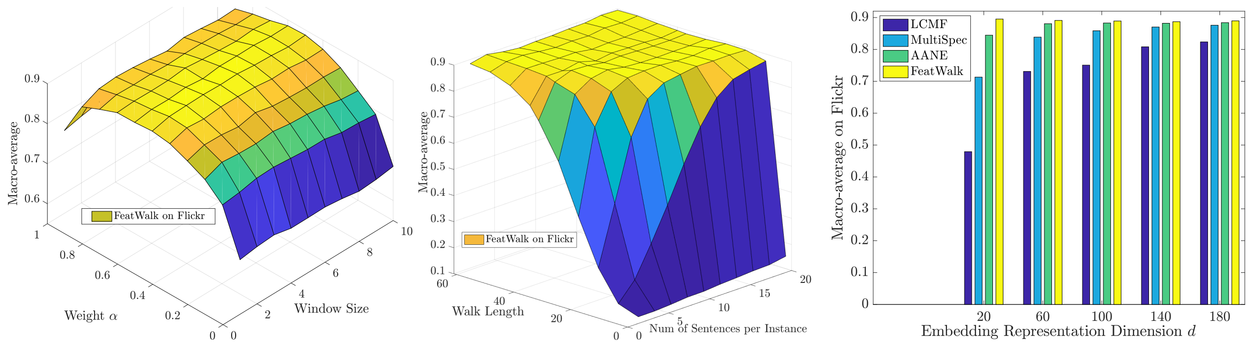
	Flickr			ACM			Yelp-sub			Yelp			
Training	25%	50%	100%	25%	50%	100%	25%	50%	100%	10%	25%	50%	100%
# Instances	3,026	4,538	7,564	19,432	29,147	48,579	19,921	29,881	49,802	69,723	99,605	149,407	249,012
NMF	0.629	0.718	0.773	0.653	0.660	0.664	0.680	0.686	0.688	0.678	0.692	0.694	0.689
Spectral	0.771	0.813	0.846	0.688	0.700	N.A.	0.683	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.
FeatWalk_A	0.803	0.841	0.868	0.676	0.675	0.667	**0.701**	0.710	0.714	0.706	0.691	0.703	0.699
DeepWalk	0.373	0.465	0.535	0.576	0.630	0.684	0.310	0.318	0.350	0.324	0.345	0.366	0.368
LINE	0.332	0.421	0.516	0.549	0.624	0.693	0.243	0.264	0.294	0.295	0.313	0.336	0.354
LCMF	0.676	0.725	0.749	0.690	0.706	N.A.	0.680	0.686	N.A.	N.A.	N.A.	N.A.	N.A.
MultiSpec	0.720	0.800	0.859	0.709	0.719	N.A.	0.667	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.
AANE	0.811	0.854	0.885	0.701	0.715	0.722	0.694	0.703	0.711	0.698	**0.709**	**0.711**	**0.714**
FeatWalk	**0.831**	**0.865**	**0.893**	**0.722**	**0.738**	**0.751**	0.700	**0.710**	**0.717**	**0.708**	0.691	0.704	0.701

Table 3.1: Classification performance of FeatWalk and baselines in terms of micro-average. [39]

3.2.5 Effectiveness of FeatWalk

To answer the second proposed question, we compare the classification performance of all methods, which are listed in Tables 3.1. From the results, we have three major findings. First, FeatWalk_A performs better than single feature embedding methods, i.e., NMF and Spectral. By taking advantage of the extra network information, FeatWalk further improves the performance. Second, by incorporating the heterogeneous information, FeatWalk outperforms all network embedding methods, i.e., DeepWalk and LINE. For example, FeatWalk achieves 9.8% of improvements than DeepWalk on ACM. Third, on all datasets except Yelp, FeatWalk achieves better performance than all heterogeneous feature embedding methods, i.e., LCMF, MultiSpec, and AANE. For example, on Flickr, FeatWalk achieves a gain of 0.9% over AANE. It demonstrates that the way that FeatWalk has used to incorporate the heterogeneous information is effective.

To study the performance of all methods w.r.t. different training set percentages, i.e., the percentage of instances in the training group (among the four folds in the cross-validation) that have been used, we vary it as $\{25\%, 50\%, 100\%\}$. The results on the four datasets are shown in Tables 3.1. From the results, we observe that the aforementioned findings hold consistently as the training set percentage increases. FeatWalk undeviatingly outperforms all baselines on all datasets except Yelp. On Yelp, the performance of FeatWalk decreases when the training set percentage increases from 10% to 25%. It is because the sequence sets reach the memory limit and an online version of word2vec is used to learn $\mathbf{H}$.



(a) α and window size (b) Walk length L and W/n (c) Representation dimension d

Figure 3.4: The impacts of parameters α, window size, walk length L, number of sentences per instance W/n, and d on FeatWalk. [39]

3.2.6 Parameter analysis

We now study the third proposed question. Performance of FeatWalk on Flickr as a function of the network weight α and window size, a function of L and W/n, and a function of d are shown in Figures 4.3a, 4.3b, and 4.3c. Results on other datasets are similar, so we omit them.

First, we vary α from 0 to 1 and the window size from 1 to 10. When $\alpha = 0$, only $\mathbf{G}$ is used to learn $\mathbf{H}$. When $\alpha = 1$, only $\mathbf{A}$ is used to learn $\mathbf{H}$. The window size denotes the maximum distance that is used to define context words in word2vec. From the results in Figure 4.3a, we observe that FeatWalk achieves the best performance when $\alpha = 0.62$, i.e., when the contributions of $\mathbf{A}$ and $\mathbf{G}$ are balanced. When α is fixed, the performance of FeatWalk keeps stable as the window size increases from 1 to 10. Second, we vary the walk length L as from 2 to 60 and the number of sentences per instance W/n from 2 to 20. From the results in Figure 4.3b, we find that the performance of FeatWalk keeps increasing as the product of L and W/n increases, and keeps stable when the product is sufficiently large. Third, we vary d as $\{20, 60, 100, 140, 180\}$. From the results in Figure 4.3c, we observe that, as d increases from 20 to 180, the performance of FeatWalk keeps stable and is always better than all baselines.

4. GRAPH NEURAL NETWORKS BASED ATTRIBUTED NETWORK EMBEDDING[1]

Among numerous network analysis techniques, deep network embedding architectures especially graph convolutional networks (GCN) have attracted wide attention [46, 10]. The superior empirical performance and their flexible parametrized architectures motivate us to couple them with random walks to conduct more effective node representation learning on attributed networks. Recent studies [26, 110] have attempted to exploit GCN to incorporate random walks into deep network embedding. However, they only adopt conventional random walks and the aggregation operation in GCN does not take the node order information into consideration [45].

In this chapter, we aim to perform effective random walks on attributed networks and convolve the extracted information via deep learning techniques for node representation learning. Besides the three challenges introduced in Chapter 1, there are two more challenges. First, node attributes are heterogeneous with respect to the topological structures [36]. Simply plugging them into existing random-walk-based models might burden the computation while downgrading the performance. Second, to fully take advantage of the structured walking sequences learned from attributed networks, an embedding algorithm that accords with walking would be needed. Thus, we aim to tailor deep embedding architectures towards coupling with attributed random walks and preserving the nice walking properties.

4.1 Problem Statement

The notations and definition of attributed network embedding are the same as Chapter 2. Given the nice properties of random walks such as being robust to sparsity and in line with the spectral segmentation, we propose to take advantage of random walks to boost ANE.

Definition 4.1. (Random-walk-based Attributed Network Embedding) The goal is to develop a framework that accords with the data characteristics of attributed network embedding, including

[1]This chapter is reprinted with permission from "Graph Recurrent Networks with Attributed Random Walks" by Xiao Huang, Qingquan Song, Yuening Li, and Xia Hu, 2019, Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Pages 732–740, Copyright 2019 by Association for Computing Machinery.

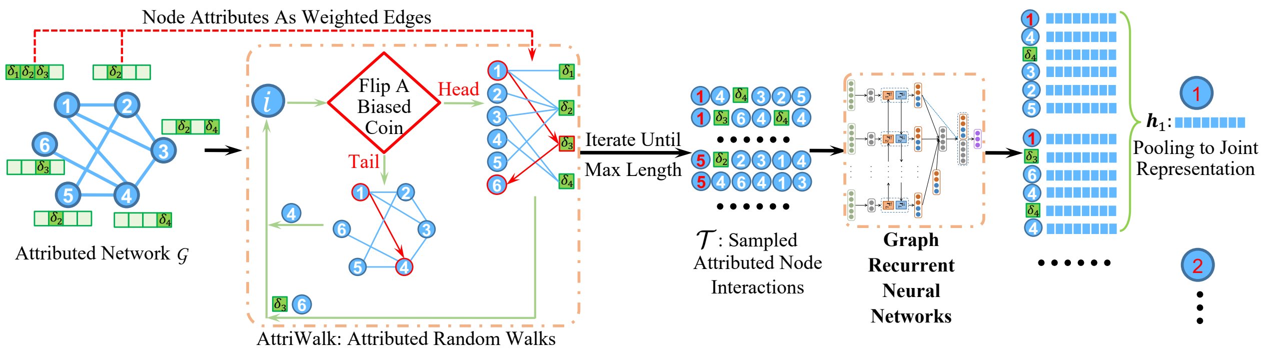

Node Attributes As Weighted Edges
Attributed Network $\mathcal{G}$
Flip A Biased Coin
Head
Tail
Iterate Until Max Length
AttriWalk: Attributed Random Walks
$\mathcal{T}$: Sampled Attributed Node Interactions
Graph Recurrent Neural Networks
h_1:
Pooling to Joint Representation

Figure 4.1: GraphRNA defines a unified walking mechanism AttriWalk, to enable joint random walks within network and node attributes, and a novel graph recurrent neural architecture to simulate attributed node interactions within the model. [38]

complex node interactions, nonlinear correlations, and heterogeneous information sources, while maintaining the nice properties brought by random walks.

4.2 Attributed Walks Based Embedding - GraphRNA

To effectively perform random-walk-based attributed network embedding, we propose a novel deep embedding framework named GraphRNA. The core idea is to perform joint random walks on attributed networks to model the attributed node interactions, and involve a recurrent neural architecture to embed the nonlinear correlations. Figure 4.1 illustrates the details of GraphRNA. We roughly separate this unified framework into three components to introduce. First, we define a unified walking mechanism to sample the complex attributed node interactions. As illustrated in Figure 4.1, to enable joint walks within the network and node attributes, we construct a bipartite network $\mathcal{A}$ based on node attributes. Nodes would have some probability of jumping to attribute categories, which increases the diversity of the walks. As a result, we convert the complex attributed node interactions into a series of informative node index sequences $\mathcal{T}$. Second, we develop an effective deep architecture named graph recurrent networks to perform embedding based on $\mathcal{T}$. It allows node representations to interact within the model in the same way as nodes interact in the original network $\mathcal{G}$. We learn a d-dimensional vector representation for each node index in each sequence in $\mathcal{T}$. Third, for each node $i \in \mathcal{V}$, we sample a small number of sequences that all take i as the starting node, denoted as set τ_i. To compute the final embedding representation of i, we apply a pooling method to fuse all vector representations of node indices in τ_i.

4.2.1 Attributed random walks - AttriWalk

We investigate to perform random walks on attributed networks, in which nodes are not only characterized by network interactions $\mathcal{G}$, but also the rich auxiliary information described by node attributes $\mathbf{A}$ [35, 59]. Jointly sampling $\mathcal{G}$ and $\mathbf{A}$ would make random walks more informative [86].

Random walks would serve as a bridge that helps our proposed deep embedding architecture handle the geometric structure. Deep neural networks have been demonstrated to be effective in modeling many types of unstructured data such as natural language, images, and audios [50]. However, when directly applied to model topological structures, existing deep architectures often achieve suboptimal performance, given that networks are in irregular or non-Euclidean spaces [15]. While attempts have been made based on graph convolutional networks [45, 46] or sophisticated objective functions [22, 51, 60, 59, 115], we explore to leverage random walks to convert the irregular attributed networks $\mathcal{G}$ into structured data. However, it is a challenging task since node attributes are within a distinct data structure than the network.

4.2.1.1 Unified walking mechanism

To tackle the heterogeneous information and effectively sample the attributed node interactions, AttriWalk defines a unified walking mechanism. The key idea is to construct a bipartite network $\mathcal{A}$ based on node attributes, and leverage it to propel the random walks more diverse and alleviate the tendency of converging to nodes with high network centralities. We now describe it in detail.

To balance the elements in $\mathbf{G}$ and $\mathbf{A}$, we first employ ℓ_1 norm to normalize each row of $\mathbf{G}$ and $\mathbf{A}$ respectively, and get $\bar{\mathbf{G}}$ and $\bar{\mathbf{A}}$. Then we collect all the m node attribute categories as a set, denoted as $\mathcal{U}$. By considering each node attribute category $\delta_k \in \mathcal{U}$ as a node, we could define a new bipartite network as $\mathcal{A} \triangleq (\mathcal{V}, \mathcal{U}, \mathcal{E})$, where $\mathcal{E}$ represents the corresponding edge set. There exists an edge between nodes $i \in \mathcal{V}$ and $\delta_k \in \mathcal{U}$, if node i contains attributes δ_k. The weight of this edge is defined as $\bar{a}_{ik}$, i.e., the value in the i^{th} row and k^{th} column of $\bar{\mathbf{A}}$.

The proposed Attriwalk would jump among all these $n + m$ nodes. Assume that currently we have jumped to a node $i \in \mathcal{V}$. As illustrated in Figure 4.1, to determine the next transition, we flip a biased coin that produces heads with probability α.

(i). If it yields head, then we walk one step on $\bar{\mathbf{G}}$. We jump to a node $j \in \mathcal{V}$, with a probability defined as,

$$P(i \to j) = \frac{\bar{g}_{ij}}{\sum_{p=1}^{n} \bar{g}_{ip}}. \tag{4.1}$$

(ii). If it turns tail, then we walk two steps on $\mathcal{A}$:

First, we jump to a node $\delta_k \in \mathcal{U}$ with a probability,

$$P(i \to \delta_k) = \frac{\bar{a}_{ik}}{\sum_{p=1}^{m} \bar{a}_{ip}} = \bar{a}_{ik}. \tag{4.2}$$

Second, from the node δ_k, we jump to a node $j \in \mathcal{V}$, with a probability defined as,

$$P(\delta_k \to j) = \frac{\bar{a}_{jk}}{\sum_{q=1}^{n} \bar{a}_{qk}}. \tag{4.3}$$

The walks on $\mathcal{A}$ enhance the diversity and flexibility of AttriWalk. In the walking, nodes in $\mathcal{V}$ could interact with each other not only based on the network $\bar{\mathbf{G}}$, but also through node attribute categories $\mathcal{U}$. When $\alpha = 1$, AttriWalk yields to vanilla random walks on the network. As α decreases, the walks would be more dependent on node attributes. The corresponding transition probability matrix could be written as,

$$\mathbf{P} = \begin{bmatrix} \alpha\bar{\mathbf{G}} & (1-\alpha)\bar{\mathbf{A}} \\ (1-\alpha)\bar{\mathbf{A}}^\top & \mathbf{0} \end{bmatrix} \in \mathbb{R}_+^{(n+m)\times(n+m)}. \tag{4.4}$$

It should be noted that the sum of each row of $\mathbf{P}$ is 1.

4.2.1.2 *Theoretical properties*

AttriWalk defines unified walks within networks $\bar{\mathrm{G}}$ and $\mathcal{A}$. The walks on $\bar{\mathrm{G}}$ inherit the nice properties of traditional random walks [68, 80, 101], including being in line with the normalized cut and spectral segmentation. The walks on $\mathcal{A}$ also follow a symmetric node similarity matrix, which could be considered as a new network, as described in Theorem 3.1.

We could use the alias method [17] to sample from a discrete probability distribution, and need to walk $\mathcal{O}(n)$ steps in total. Comparing with the time complexity of the intuitive solution $\mathcal{O}(n\mathcal{N}_A + n^2 + n)$, the time complexity of AttriWalk is $\mathcal{O}(\mathcal{N}_A + n)$.

4.2.1.3 *Sampling settings*

Based on AttriWalk, we sample the local topological structure and node attributes of all nodes. We employ the settings as follows. All the random walks would be truncated to length L. For each node $i \in \mathcal{V}$, we conduct a fixed number B of fixed-length walks. They all use i as the initial node. We denote all the B learned node index sequences as a set τ_i. We collect the learned sequences of all nodes as the set $\mathcal{T}$. For example, in Figure 4.1, we have $B = 2$, $L = 6$, and set $\tau_1 = \{\{1, 4, \delta_4, 3, 2, 5\}, \{1, \delta_3, 6, 4, \delta_4, 4\}\}$. In such a way, we convert the complex attributed node interactions into a series of informative node index sequences $\mathcal{T}$.

4.2.2 Graph recurrent networks - GRN

We now study to perform effective embedding by taking advantage of the learned sequences $\mathcal{T}$. A widely adopted approach [80, 24] is to consider the sequences as sentences and nodes as words, and apply word embedding techniques [69] to learn a latent representation for each word. This approach has been demonstrated to be effective in plain networks. However, in attributed networks, node properties are also affected by node attributes. Word embedding models could not take this auxiliary information into consideration. Another line of work [26, 110] exploited to leverage graph convolutional networks (GCN) to incorporate the random walks into deep learning models. But the aggregation operation in GCN does not take the node order information into consideration [45, 46].

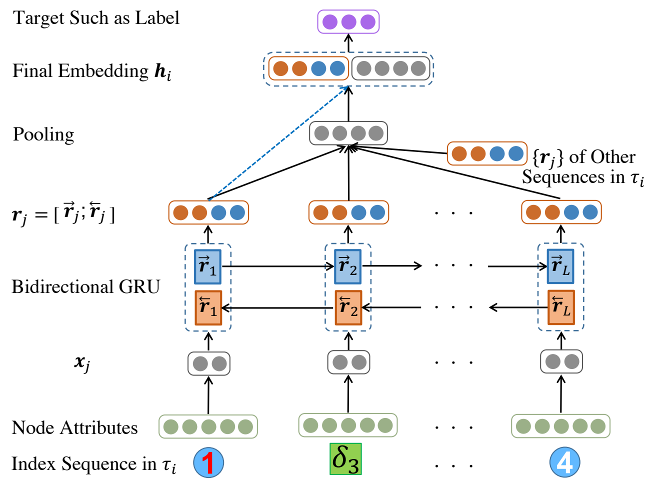

Target Such as Label
Final Embedding h_i
Pooling
$\{r_j\}$ of Other Sequences in τ_i
$r_j = [\vec{r}_j; \overleftarrow{r}_j]$
Bidirectional GRU
x_j
Node Attributes
Index Sequence in τ_i
1
δ_3
4

Figure 4.2: Architecture of the designed tailored graph recurrent networks. [38]

4.2.2.1 Proposed neural architecture

The sequences in $\mathcal{T}$ describe how nodes interact with neighbors through the topological structure and node attributes. The hidden state sequences in recurrent neural networks (RNN) naturally accord with these sampled node interactions. Thus, we explore to utilize RNN to model the order information in $\mathcal{T}$. The architecture of the proposed graph recurrent networks is illustrated in Figure 4.2, with details as follows.

I. Let $\{1, \delta_3, 6, 4, \delta_4, 4\}$ be a length L sequence in τ_i. We map each of the L indices into a m-dimensional vector. If it is the index of a node $j \in \mathcal{V}$, then we map it to its node attributes $\mathbf{a}_j$. If it is the index of attribute category $\delta_j \in \mathcal{U}$, then we map it to a one-hot vector $\mathbf{e}_j$ with the j^{th} element equals to 1.

II. We employ a fully connected layer to reduce the dimension of node attributes, and get vectors $\{\mathbf{x}_j\}$, for $j = 1, \ldots, L$. Mathematically, $\mathbf{x}_j$ is computed based on,

$$\mathbf{x}_j = \sigma(\mathbf{a}_j \mathbf{W}_a + \mathbf{b}_a), \text{ or } \mathbf{x}_j = \sigma(\mathbf{e}_j \mathbf{W}_a + \mathbf{b}_a), \tag{4.5}$$

where σ is the sigmoid function, and $\mathbf{W}_a \in \mathbb{R}^{m \times d}$ is the weight matrix. Each of its row $\mathbf{w}_j$ is corresponding to a latent representation of the node attribute category δ_j. Thus, when there is an index δ_j in the input sequence, $\mathbf{e}_j$ would help it lookup $\mathbf{w}_j$.

III. Given $\{\mathbf{x}_j\}$, we use a bidirectional RNN such as long short-term memory and gated recurrent units [6] to learn a forward hidden state sequence $(\overrightarrow{\mathbf{r}}_1, \overrightarrow{\mathbf{r}}_2, \ldots, \overrightarrow{\mathbf{r}}_L)$ and a backward hidden state sequence $(\overleftarrow{\mathbf{r}}_1, \overleftarrow{\mathbf{r}}_2, \ldots, \overleftarrow{\mathbf{r}}_L)$. Each hidden state could be interpreted as a node, and the RNN enables nodes to interact with both the forward and backward neighbors. It is in line with the way nodes interact in the original network $\mathcal{G}$. In addition, the corresponding node attributes have been transmitted into each hidden state, which empowers the heterogeneous network and node attribute information incorporate within a natural manner.

We take the forward hidden state sequence $\{\overrightarrow{\mathbf{r}}_j\}$ as an example. $\overrightarrow{\mathbf{r}}_j$ is calculated based on,

$$\mathbf{f}_j = \sigma(\mathbf{x}_j\mathbf{W}_{xf} + \overrightarrow{\mathbf{r}}_{j-1}\mathbf{W}_{hf} + \mathbf{b}_f), \tag{4.6}$$

$$\mathbf{i}_j = \sigma(\mathbf{x}_j\mathbf{W}_{xi} + \overrightarrow{\mathbf{r}}_{j-1}\mathbf{W}_{hi} + \mathbf{b}_i), \tag{4.7}$$

$$\mathbf{o}_j = \sigma(\mathbf{x}_j\mathbf{W}_{xo} + \overrightarrow{\mathbf{r}}_{j-1}\mathbf{W}_{ho} + \mathbf{b}_o), \tag{4.8}$$

$$\mathbf{c}_j = \mathbf{f}_j \circ \mathbf{c}_{j-1} + \mathbf{i}_j\text{tanh}(\mathbf{x}_j\mathbf{W}_{xc} + \overrightarrow{\mathbf{r}}_{j-1}\mathbf{W}_{hc} + \mathbf{b}_c), \tag{4.9}$$

$$\overrightarrow{\mathbf{r}}_j = \mathbf{o}_j \circ \text{tanh}(\mathbf{c}_j). \tag{4.10}$$

$\mathbf{f}_j$, $\mathbf{i}_j$, and $\mathbf{o}_j$ denote the forget, input, and output gates' activation vectors. $\mathbf{c}_j$ represents the cell state vector. $\circ$ denotes the Hadamard product. tanh denotes the Hyperbolic tangent function. The output of this bidirectional RNN layer is obtained by concatenating the forward and backward hidden state vectors, i.e., $\mathbf{r}_j = [\overrightarrow{\mathbf{r}}_j, \overleftarrow{\mathbf{r}}_j]$.

IV. For each node $i \in \mathcal{V}$, there are totally BL indices in its sequence set τ_i. Correspondingly, we could get BL embedding vectors $\{\mathbf{r}_j\}$ from the bidirectional RNN layer. We follow a well studied architecture [26, 110] to compute $\mathbf{h}_i$ the final embedding representation of node i. We first apply a pooling method to combine all the B sequences into one, denoted as $\{\bar{\mathbf{r}}_1, \bar{\mathbf{r}}_2, \ldots, \bar{\mathbf{r}}_L\}$. Then we apply a pooling method again to merge all the $\{\bar{\mathbf{r}}_2, \bar{\mathbf{r}}_3, \ldots, \bar{\mathbf{r}}_L\}$ into $\hat{\mathbf{r}}_i$. The final embedding representation $\mathbf{h}_i$ is defined as,

$$\mathbf{h}_i = [\hat{\mathbf{r}}_i, \bar{\mathbf{r}}_1]. \tag{4.11}$$

It should be noted that sequences in τ_i must start with index i. Thus, $\bar{\mathbf{r}}_1$ is the latent representation of node i before the second pooling.

4.2.2.2 *Relations with graph convolutional networks*

The key idea of GCN is to input node attributes or a trainable embedding lookup table into a special multilayer perceptron, in which each node's representation is learned via averaging its neighbors' representations in the previous layer [26, 45]. It could be considered as a first-order approximation of spectral graph convolutions [15, 46].

GCN could be interpreted as a special case of our proposed graph recurrent networks (GRN). If we remove the bidirectional RNN layer in GRN, as the number of sampling B keeps increasing, the pooling operation in GRN would approach to inductively learning from all neighbors. Several efforts [11, 110] also have demonstrated that random sampling from multi-hop neighbors would improve the performance of GCN. The attributed random walks enable us to boost GCN via taking the node interaction order information into consideration. While GCN achieves the node interactions via layer to layer, GRN empowers nodes and attribute categories to interact within the same bidirectional RNN layer. GCN is not robust to multiple layers since it would become over-smoothing [58]. Thus, nodes could only interact a few times within GCN. But in GRN, the bidirectional RNN layer allows much longer interactions.

4.2.3 Optimization

GraphRNA could be trained with an unsupervised, supervised, or semisupervised setting. This nice property is inherit from GCN [26, 46]. The loss function is not the focus of this paper. We take a supervised setting as an example. Based on the cross-entropy error, the objective function could be defined as follows,

$$\mathcal{L} = -\sum_{i \in \mathcal{V}} \mathbf{y}_i^\top \log(\text{softmax}(\sigma(\mathbf{h}_i \mathbf{W}_h + \mathbf{b}_h))). \tag{4.12}$$

where $\mathbf{y}_i$ is a one-hot vector denoting the label of node i. It is straightforward to replace $\mathcal{L}$ by other task-specific objective functions such as the negative sampling based unsupervised objective

```
Algorithm 2: GraphRNA [38]
  Input: $\mathbf{G}$, $\mathbf{A}$, $d$, $B$, $L$, $\alpha$, and labels.
  Output: Attributed network embedding representation $\mathbf{H}$.
1 $\bar{\mathbf{G}}$ and $\bar{\mathbf{A}} \leftarrow$ use $\ell_1$ norm to normalize each row of $\mathbf{G}$ and $\mathbf{A}$;
2 Compute $\mathbf{P}$ based on Eq. (4.4);
3 Construct a probability table and an alias table for each $\mathbf{p}_i$;
4 for Node $i$ in $\mathcal{V}$ do
5     for $j = 1 : B$ do
6         Set $i$ as the initial node;
7         Perform a $(L-1)$ length attributed random walks based on the probability tables
            and alias tables;
8         Append the learn sequence to $\tau_i$;
9     end
10    Append $\tau_i$ to $\mathcal{T}$;
11 end
12 for $epoch = 1 : epoch_{max}$ do
13    Shuffle the order of $\tau_i$ in $\mathcal{T}$;
14    for Set $\tau_i$ in $\mathcal{T}$ do
15        Take all $B$ sequences in $\tau_i$ as input and node $i$'s label as output to train the GRN, as
            shown in Figure 4.2;
16        Update weight matrices and bias terms to minimize $\mathcal{L}$ in Eq. (4.12);
17    end
18 end
19 for $\tau_i = \tau_1 : \tau_n$ do
20    $\mathbf{h}_i \leftarrow$ Input all $B$ sequences in $\tau_i$ into the trained GRN;
21 end
```

function [26, 91].

We summarize the optimization of GraphRNA in Algorithm 2. Given an attributed network $\mathcal{G} = (\mathcal{V}, \mathbf{G}, \mathbf{A})$, we first normalize $\mathbf{G}$ and $\mathbf{A}$ to construct $\mathbf{P}$. It has summarized the transition probabilities of the joint random walks within $\mathbf{G}$ and $\mathbf{A}$. We employ the alias method [17] to sample truncated random walks from the discrete probability distribution in $\mathbf{P}$. For each node $i \in \mathcal{V}$, we sample B sequences and append them to τ_i. They all take i as the starting node. We use τ_i and node i's label to train the GRN, based on the objective function $\mathcal{L}$ in Eq. (4.12). After training the GRN over several epochs, we use it to embed τ_i into $\mathbf{h}_i$.

4.3 Experiments

In this section, we analyze the proposed framework GraphRNA empirically on three real-world attributed networks. Through the experiments, we target at answering three research questions. (1) How effective is the embedding representations learned by GraphRNA compared with the ones learned by state-of-the-art ANE methods, in terms of node classification? (2) GraphRNA mainly consists of attributed random walks and GRN. How much does each component contribute? (3) What is the impact of the parameters on GraphRNA, including the probability α, number of sequence per node B, sequence length L, and embedding dimension d?

We employ three publicly available attributed networks in the experiments, including BlogCatalog, Flickr, ACM. They have been introduced in Chapters 2 and 3.

4.3.1 Baseline methods

To validate the effectiveness of GraphRNA, we include three categories of baselines. First, to study how informative is each single source, we include two network embedding methods, i.e., DeepWalk and LINE, and one node attribute embedding method Attribute-Spec. Second, to investigate how effective is GraphRNA compared with shallow models, we include two state-of-the-art shallow ANE methods, i.e., MultiSpec and AANE. Third, to analyze how much GraphRNA has advanced the GCN architectures, we include the vanilla GCN and GraphSAGE. Several baselines have been introduced in Chapter 2, including DeepWalk, LINE, MultiSpec, and AANE.

- *Attribute-Spec [101]:* It calculates the similarities between each pair of nodes based on $\mathbf{A}$ to construct a new graph, and learns $\mathbf{H}$ from it via spectral embedding.
- *GCN [46]:* The vanilla GCN. It learns $\mathbf{H}$ based on the first-order approximation of spectral graph convolutions.
- *GraphSAGE [26]:* It advances GCN via introducing a set of aggregator functions that learn to aggregate node attributes from a node's local neighbors.

Training set for evaluation		BlogCatalog 25%	BlogCatalog 50%	BlogCatalog 100%	Flickr 25%	Flickr 50%	Flickr 100%	ACM 25%	ACM 50%	ACM 100%
Single Source	DeepWalk	0.551	0.611	0.672	0.373	0.465	0.535	0.576	0.630	0.684
	LINE	0.545	0.624	0.684	0.332	0.421	0.516	0.549	0.624	0.693
	Attribute-Spec	0.791	0.841	0.869	0.771	0.813	0.846	0.688	0.700	0.704
Shallow ANE	MultiSpec	0.788	0.849	0.896	0.720	0.800	0.859	0.709	0.719	N.A.
	AANE	0.878	0.913	0.932	**0.811**	0.854	0.885	0.701	0.715	0.722
Deep ANE	GCN-hidden	0.657	0.701	0.734	0.521	0.558	0.588	0.754	0.766	0.789
	GraphSAGE-hidden	0.801	0.876	0.917	0.612	0.743	0.841	0.727	0.769	0.785
	GraphRNA-hidden	0.857	0.903	0.938	0.705	0.817	0.881	0.727	0.755	0.781
End-to-End	GCN	0.683	0.716	0.740	0.514	0.563	0.590	**0.763**	**0.776**	**0.798**
	GraphSAGE	0.820	0.890	0.924	0.626	0.757	0.852	0.741	0.769	0.785
	GraphRNA-noAttriW	0.865	0.909	0.942	0.645	0.777	0.874	0.746	0.768	0.788
	GraphRNA-noRNN	0.874	0.920	0.943	0.769	0.855	0.902	0.739	0.768	0.784
	GraphRNA	**0.885**	**0.925**	**0.948**	0.776	**0.866**	**0.905**	0.747	0.768	0.790

Table 4.1: Classification performance of all methods on all datasets in terms of micro-average. [38]

4.3.2 Experimental settings

We use the same experimental settings as Chapter 2. The three real-world datasets that we employed in the experiments, including BlogCatalog, Flickr, and ACM, have been introduced in Chapters 2 and 3. For all the deep models, except using their learned $\mathbf{H}$ to train an SVM classifier, we also combine them with the multilayer perceptron classifier into end-to-end models to perform the classification. For all methods, we use the validation set to fine tune the hyperparameters, as well as conduct the early stopping in deep models. If it is not specified, all the nodes within the training set are used to train the classifier. d is set as 100 in default. All the reported experimental results are the arithmetic average of ten runs.

4.3.3 Effectiveness evaluation

We now answer the three questions proposed at the beginning of this section one by one. For the first question, we compare GraphRNA with all three categories of baselines.

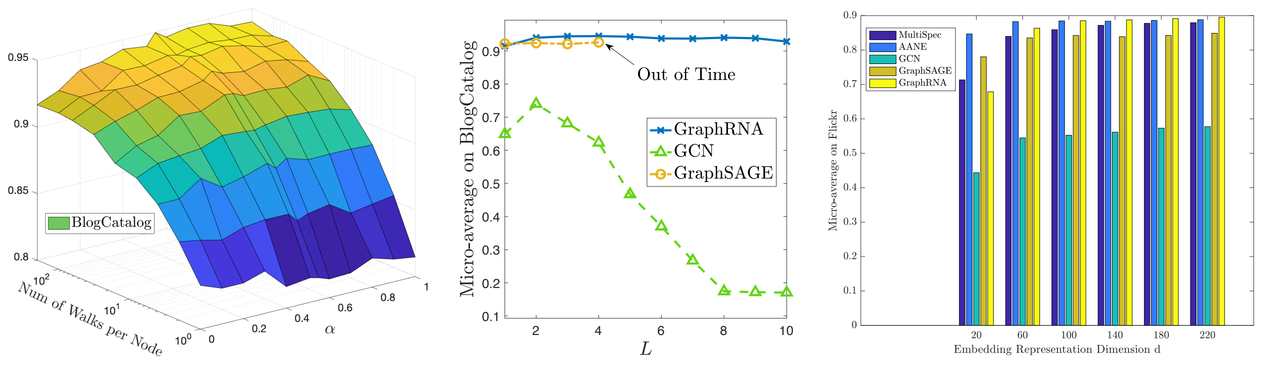
For all the deep baselines and GraphRNA, labels are used to train the models, and the last hidden state is returned as $\mathbf{H}$. We also train them together with multilayer perceptron classifiers, and report the results in the end-to-end method category. The experimental classification results in terms of micro-average on the three datasets are summarized in Table 4.1. Similar results are observed based on macro-average scores, so we omit them.

From the results in Table 4.1, we have three major observations. First, GraphRNA achieves the best performance on datasets BlogCatalog and Flickr, in terms of micro-average. On ACM, it also achieves a comparable performance as the best one. For example, on BlogCatalog, GraphRNA achieves 28.1% of improvements than GCN and 2.6% of improvements than GraphSAGE. GCN performs well on citation networks, but badly on social networks, i.e., BlogCatalog and Flickr, while GraphRNA achieves good performance consistently. Second, in general, ANE methods performs better than the methods with a single source, and deep ANE models achieve higher micro-average scores than shallow ones. For instance, GraphRNA achieves a gain of 12.3% over MultiSpec on BlogCatlog and a gain of 2.3% over AANE on Flickr. Third, for all the deep baselines and GraphRNA, their performance has been slightly improved by the end-to-end training settings, comparing with using hidden states to train SVM classifiers separately. For example, on BlogCatalog, GraphSAGE achieves 1.1% of improvements than GraphSAGE-hidden. It could be explained by the fact that the representation learning and classification are trained uniformly in end-to-end models.

For each dataset, we vary the percentage of the training set that has been used for learning the classifiers as $\{25\%, 50\%, 100\%\}$. It means that only $\{25\%, 50\%, 100\%\}$ of the four folds selected by the five-fold cross-validation, have been used. From the results in Table 4.1, we have similar observations as above, as the training percentage varies. For example, when the training percentage is set as 50% on Flickr, GraphRNA achieves 14.4% improvement compared with GraphSAGE. As the training percentage increases, the performance of all methods increases.

4.3.4 Component contribution analysis

We now investigate the second question, i..e, how much could GraphRNA's two major components, i.e., AttriWalk and GRN contribute. Two variations of GraphRNA are used for this ablation study. Their node classification performance on all datasets is included in Table 4.1.



(a) α and B (b) Interaction depth L (c) Representation dimension d

Figure 4.3: The impacts of parameters the probability α, the number of walks per node B, the length of each truncated random walk L, and d on GraphRNA. [38]

- *GraphRNA-noAttriW:* GraphRNA without using attributed random walks to sample $\mathcal{T}$. Instead, it employs the vanilla random walks to learn $\mathcal{T}$.
- *GraphRNA-noRNN:* GraphRNA with the bidirectional RNN layer removed.

From the results in Table 4.1, we have three major observations as follows. First, without AttriWalk, the performance of GraphRNA-noAttriW decreases, especially on Flickr. It demonstrates the effectiveness of the component GRN. It should be noted that node attributes are still available for GraphRNA-noAttriW. It outperforms GraphSAGE on all datasets. For example, it achieves 1.9% of improvements than GraphSAGE on BlogCatalog. The major difference between GraphRNA-noAttriW and GraphSAGE is that the former one could take the order information in the learned sequences into consideration via GRN. This validates the effectiveness of GRN. Second, without the RNN layer, GraphRNA-noRNN achieves slightly worse performance than GraphRNA. For instance, GraphRNA achieves 0.3% of improvements than GraphRNA-noRNN on Flickr. It validates the effectiveness of the proposed GRN architecture. Third, AttriWalk would bring more performance improvements than the GRN, which indicates that AttriWalk might be a more important component. In the experiments, the RNN in GRN is instantiated by the long short-term memory. A tailored or more sophisticated RNN architecture might further improve the performance of GraphRNA [42].

4.3.5 Parameter sensitivity analysis

We now study the impact of parameters α, B, L, and d on GraphRNA. α denotes the probability of conducting vanilla random walks within the joint walks AttriWalk. B denotes the number of sequences that we have sampled per nodes. We plot the performance on BlogCatalog as a function of α and B in logarithmic scale, with results shown in Figures 4.3a. Similar results are observed on other datasets, so we omit them. Form the results in Figures 4.3a, we observe that the performance of GraphRNA increases as B increases from 1 to 100, and then keeps stable. GraphRNA achieves the best performance when α is around 0.5, and then achieves slightly worse results when $\alpha = 0$ and 1. It should be noted that, when $\alpha = 0$, GraphRNA walks purely on node attributes, so the network information is no longer available. When $\alpha = 1$, both the network and node attribute information is still available, except that AttriWalk becomes vanilla random walks.

L denotes the length of each truncated sequence. It could be interpreted as the depth that nodes could interact. Larger L means that nodes have more opportunities to reach multi-hop neighbors. While GraphRNA allows nodes to interact within the same RNN layer, in GCN and GraphSAGE, nodes would interact from layer to layer. We vary L from 1 to 10, and the changes of GraphRNA's performance is shown as a blue curve in Figures 4.3b. By interpreting L as the depth of node interaction, i.e., number of layers in GCN and GraphSAGE, we plot the performance of GCN and GraphSAGE as the function of L. The results are shown as orange and green curves in Figures 4.3b. From the results, we observe that both GraphRNA and GraphSAGE allow a much larger number of interactions than the villain GCN.

In GraphRNA, the representation dimension d is interpreted as the number of hidden units in the layer before last target layer, as shown in Figure 4.2. We vary it from 20 to 220, and the performance of all ANE methods is shown in Figures 4.3c. From the results, we observe that when d is too small such as 20, the performance of GraphRNA would be limited. Similar observations are made on all deep models. Their performance becomes stable when d is large enough.

5. EXPLORING EXPERT COGNITION FOR EMBEDDING[1]

We have introduced a series of attributed network embedding frameworks, including AANE, FeatWalk, and GraphRNA. They have different theoretical foundations, and could serve as an infrastructure for developing various analysis tasks. Recent advances in explanation-based learning and human-in-the-loop models show that by involving experts, the performance of many learning tasks can be enhanced. It is because experts have a better cognition in the latent information such as domain knowledge, conventions, and hidden relations. It motivates us to employ experts to transform their meaningful cognition into concrete data to advance network embedding. In this Chapter, we study a novel problem of exploring expert cognition for attributed network embedding and propose a principled framework NEEC.

5.1 Introduction

The helpfulness of expert cognition motivates us to investigate the potential of utilizing it to learn more informative embedding representations. We define the *expert cognition* as the intelligence-related information that experts know beyond the data, such as the understanding of domain knowledge [106], the awareness of conventions [14], and the perception of latent relations. Existing work such as explanation-based learning [16] and human-in-the-loop models [31, 44] suggest that the involvement of experts could increase the performance of many learning tasks. Expert cognition plays an essential role in advancing these data analysis. Since most existing network embedding algorithms are data-driven, we are motivated to explore how to take advantage of the expert cognition to enhance the performance of embedding. Sentiment analysis is a typical example of the utilization of expert cognition, which has been successfully applied to improve the performance of different recommendations [114]. In product review platforms such as Epinions and Slashdot, items are recommended to users based on their historical reviews. It has been shown

[1]This chapter is reprinted with permission from "Exploring Expert Cognition for Attributed Network Embedding" by Xiao Huang, Qingquan Song, Jundong Li, and Xia Hu, 2018, Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, Pages 270–278, Copyright 2018 by Association for Computing Machinery.

that most reviews convey different levels of sentiment polarization [32], and the expert-created sentiment lexicons such as MPQA and SentiWordNet [87] are widely used to advance the review analysis and recommendation [18, 114].

Although some efforts have been devoted to directly learning expert cognition from the existing human-generated data [18], such concrete related data is still often limited and could be in different forms. Motivated by the success of active learning [85, 84] and interactive data mining [1, 31], in this paper, we propose to explore expert cognition via actively querying the experts. We aim to convert their abstract cognition into concrete answers, and incorporate them into attributed network embedding towards a more informative low-dimensional representation.

However, actively exploring expert cognition is a nontrivial task, with three major challenges as follows. First, expert cognition does not have a concrete format and is not easy to be measured, since it is related to the intelligence. Traditional solutions are to design specific algorithms according to the experts' understanding [16, 44]. It is hard to be generalized as they involve enormous engineering experimentations to transform the domain knowledge to algorithms. Second, in a real-world system, there are usually various types of expert cognition such as the comprehension of word meaning and the discernment of missing edges [31]. It is difficult to model all of them, or identify the types that can lead a significant performance gain in the embedding within a limited number of trials. It is also expensive to design specific models for different attributed networks and different embedding algorithms. A general way of modeling the expert cognition is essential. Third, the expert cognition is laborious to obtain as it involves humans in the whole process. Given a limited amount of human effort, we aim to design the queries in an effective way to maximize the total amount of learned expert cognition. Meanwhile, the returned answers from the experts could be heterogeneous with the attributed network. It is desired to have answers that could be easily incorporated into the embedding.

To bridge the gap, we formally define the problem of exploring expert cognition for attributed network embedding and study two research questions. (1) How to design effective, general, and concise format of queries to save the experts' efforts? (2) How to select the most meaningful queries to maximize the total amount of learned expert cognition, given a limited amount of human effort? (3) How to quantify the amount of expert cognition that contained in the returned answers?

5.2 Problem Statement

The notations and definition of attributed network embedding are the same as Chapter 2. Suppose the raw data of $\mathbf{A}$ is human-readable such as the paper abstract or the product review. For ease of presentation, in this paper, we focus on the undirected networks and propose a general framework. It can also be directly applied to the directed networks.

Definition 5.1. (Expert Cognition) For real-world attributed networks, the domain experts often have a better comprehension beyond the data, such as the understanding of domain knowledge, the awareness of conventions, and the perception of latent relations. We refer their knowledge on this type of abstract, meaningful, and intelligence-related information as the expert cognition.

Instead of directly modeling expert cognition, we propose a novel way to learn it, i.e., querying the experts a number of trials to model their abstract cognition as concrete answers. The existing related data is often limited, and it is difficult to design specific models for different systems as the given data could be in various forms. Thus, we focus on actively learning from the experts. It is motivated by the success of active learning [85], which has been shown to be helpful in reducing the efforts of labeling. The key idea is, if we are allowed to select the nodes for training, a comparable classification accuracy could be achieved even with fewer training labels. However, existing methods cannot be applied to our problem since it is often expensive to obtain concrete labels, such as the labels in ranking systems or community detection problems. Also, it is challenging to incorporate labels into the embedding since they are heterogeneous to the attributed network.

We aim to actively explore the expert cognition in a more general way. We refer experts as the oracle and formally define the problem of exploring expert cognition for attributed network embedding as follows.

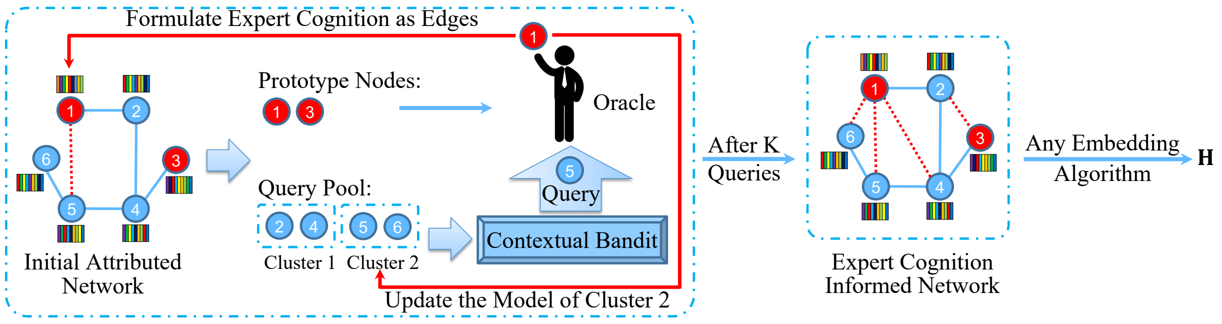

Formulate Expert Cognition as Edges
Prototype Nodes:
Oracle
Query Pool:
Query
Contextual Bandit
Cluster 1
Cluster 2
Update the Model of Cluster 2
Initial Attributed Network
After K Queries
Expert Cognition Informed Network
Any Embedding Algorithm
H

Figure 5.1: Illustration of the proposed Network Embedding with Expert Cognition - NEEC. [37]

Given an attributed network $\mathbf{G}$ associated with node attributes $\mathbf{A}$ and corresponding raw data, we aim to seek a d-dimensional vector representation $\mathbf{h}_i$ for each node i, such that both the topological structure and node attribute proximity could be well preserved in $\mathbf{H}$. Meanwhile, the system is allowed to learn from the oracle by performing K queries one by one. Given a limited amount of effort the oracle could devote, we aim to design the queries in a general and efficient way to include as much as possible expert cognition in the K returned answers. As a result, we could incorporate the answers into the embedding towards a more informative expert cognition informed representation $\mathbf{H}$.

5.3 Embedding with Expert Cognition - NEEC

To efficiently model and assimilate the expert cognition actively learned from the oracle into attributed network embedding, we propose the Network Embedding with Expert Cognition framework - NEEC. It has three components as illustrated in Figure 5.1. (1) We design a general and concise form of queries to learn expert cognition from the oracle while greatly save his/her effort. It allows us to model the meaningful but abstract expert cognition as systematical answers. (2) We explore novel algorithms for the two steps to find the top K meaningful queries. The first step aims to find B representative and distinct nodes as prototypes, and the second step iteratively selects K nodes from the remaining nodes with the largest amount of expected learned expert cognition. (3) We formulate the returned K answers from the oracle as cognition edges and add them into the initial network structure. Figure 5.1 illustrates an example on a network of six nodes. We first select $B = 2$ nodes as prototypes (in red), and set the remaining nodes (in blue) as a query pool. In each query, we select a node i (e.g., $i = 5$) to query. The oracle needs to indicate a node from the prototypes (e.g., $j = 1$) that is the most similar to the queried node i. All these answers will be added into the network structure in the form of weighted edges, named as cognition edges (red dotted lines). In Figure 5.1, the queried nodes $i = 4, 5, 6$ are more similar to prototype node $j = 1$, so we add weighted edges between them. With these cognition edges, different attributed network embedding methods could be directly applied to the expert cognition informed network towards a more informative low-dimensional representation $\mathbf{H}$.

5.3.1 Prototype-based form of queries

Given a limited amount of effort the oracle could devote, simpler queries would allow us to have a larger number of queries K. Meanwhile, we expect each query to be more effective in exploring the expert cognition from the oracle. These two aspects often contradict each other, since complex and specific queries tend to capture more information, but are both hard to design and answer. An intuitive solution is to query the latent relations of node pairs to model the expert cognition as identified missing edges. However, it might not be efficient since the total number of queries K is quite limited compared with the total number of node pairs, i.e., $K \ll \binom{n}{2}$. It is also difficult to select the K most meaningful node pairs from all the unconnected pairs. Our proposed solution is to learn a more general type of node relations from the oracle, and in a systematical way to increase the total amount of learned expert cognition.

5.3.1.1 Learning expert cognition with prototypes

Based on the exemplar theory and prototype theory in cognitive science, we propose the prototype-based form of queries.

Theory 5.2. (Exemplar Theory) *[72] In psychology, humans cognize new stimuli by comparing with exemplars already have in memory, and classify objects based on their similarity to these exemplars.*

Theory 5.3. (Prototype Theory) *[83] For a category of stored exemplars in humans' memory, there exists an abstract average of all members called the prototype, and humans perform categorizations based on prototypes when experiencing new objects.*

Motivated by the process of humans cognizing new objects, we first select B nodes and define them as the prototypes of B categories respectively. Then we classify K other selected nodes into these B categories according to the oracle's cognition. Such classification process could capture the oracle's comprehensive comprehension of the K nodes. In the end, we formulate this valuable comprehension as cognition edges defined as follows.

Definition 5.4. (Prototype Node) We select a small number (i.e., B) of representative nodes to serve as prototypes in the oracle's memory, and refer them as prototype nodes.

Definition 5.5. (Cognition Edge) In each query, we add an edge g_{ij} with a predefined weight γ between the newly selected node i and its most similar prototype node j. We refer g_{ij} as a cognition edge. If there is already an edge between nodes i and j in the initial network, we will add the weight γ into this edge.

Figure 5.1 illustrates the basic idea of the prototype-based form of queries. Initially, nodes 1 and 3 are selected as prototype nodes. In each query, we select another new node i to query the oracle. He/she needs to determine node i is more similar to which prototype, based on his/her understanding of nodes' human-readable raw data and expertise. A cognition edge would be added to the network according to the returned answer. It models the category of node i in the oracle' cognition. After K queries, we could directly apply an embedding algorithm to the expert cognition informed network towards a more informative unified representation.

5.3.1.2 Analysis of prototype-based form of queries

We have proposed to learn the expert cognition with prototype nodes and model it as cognition edges. It enjoys several nice properties as follows. First, evaluating similarity takes a much smaller amount of human effort comparing to other technical tasks such as labeling. The prototype-based form of queries are general and simple with no specific similarity value required. Second, the data for some specific nodes might be limited, but domain experts' knowledge and cognition could help. For example, researchers can understand the main idea of papers by reading the abstracts, while these short paragraphs are often too limited to computers. Third, the oracle only needs to check $B + K$ nodes in total, which are selected from a pool with size n. It is much smaller than $\binom{n}{2}$.

5.3.2 Prototype node selection algorithm

We propose two algorithms for the prototype node selection to handle different scenarios. The key idea is to make prototype nodes representative and avoid being too similar to each other, such that the oracle could easily distinguish them.

The first algorithm is k-medoids, motivated by the work of [76]. This algorithm is designed for the network with rich initial data. It aims to select B prototype nodes that cluster the entire n nodes into B distinct groups. An updating rule that similar to the k-means algorithm is employed. It first selects B nodes randomly, and then alternatively conducts the following two proceeds until no change could be made. (1) Assign nodes to their nearest prototype nodes' clusters based on node attribute similarity. (2) For each node, compute the sum of its similarities to all other nodes within the same cluster. For each cluster, set the node that has the maximum similarity sum as the new prototype node.

The second algorithm is a random strategy for networks with limited initial data or with a large number of nodes. We first select B nodes from the entire n nodes randomly, and then keep removing the prototype nodes that are too similar to each other and replacing them with new random nodes. The similarities are based on the initial attributed network.

5.3.3 Query selection algorithm

We have proposed to learn the expert cognition by linking nodes with their most similar prototype nodes. In such a way, we formulate the query selection problem as a node selection task. The goal is to find K effective nodes to maximize the total amount of expert cognition contained in the learned cognition edges.

An intuitive solution is to greedily return the top K important nodes [71], aiming to make the learned cognition edges more meaningful. However, this might not end up with a good result. Because important nodes tend to have more edges included in the initial network [70], and the learned cognition edges would become less influential. It also fails to take other information into consideration, such as the feedback from the oracle and the node attributes.

5.3.3.1 Exploitation and exploration trade-off

To find the K effective nodes, we first formally define the problem. Let $r_{i,k}$ denote the amount of expert cognition contained in the k^{th} returned answer, referred as *reward*. Here i denotes the k^{th} node we have queried. We have no access to the reward $r_{i,k}$ before asking the k^{th} query. The goal is to maximize the total rewards after all K queries, i.e.,

$$\underset{i \in \mathcal{P}_1}{\text{maximize}} \quad \mathcal{J}_K = \sum\nolimits_{k=1}^{K} r_{i,k}, \tag{5.1}$$

where $\mathcal{P}_k$ denotes the query pool in the k^{th} iteration. In the first iteration, the initial $\mathcal{P}_1$ has $(n - B)$ nodes since we have employed B nodes as prototype nodes. For the node that is already selected previously, its reward is 0 since no new cognition edge would be added. We remove it from the pool and get the $\mathcal{P}_{k+1}$.

In the beginning, we have limited information about the relation between the node i and $r_{i,k}$. Therefore, when performing the querying, we aim to not only maximize the learned expert cognition in the current trial, but also explore the system to optimize the incoming iterations. To tackle this exploration and exploitation trade-off and maximize the total rewards $\mathcal{J}_K$, we propose a novel contextual bandit [57, 111] algorithm to perform the node selection.

5.3.3.2 Contextual bandit algorithm

Our main idea is to assume that nodes within the same cluster share the same model that measures their relations to the rewards. For nodes $i \in \mathcal{P}_1$, we cluster them into d models, and define an information vector $\mathbf{x}_i$ to describe the property of each node i. We employ a linear function to model the correlation between $\mathbf{x}_i$ and $r_{i,k}$, i.e.,

$$\mathrm{E}[r_{i,k}|\mathbf{x}_i] = \mathbf{x}_i\theta_a + \eta_a, \tag{5.2}$$

where $a \in \{1, 2, \dots, d\}$ denotes the cluster that node i belongs to, and η_a is a random noise with Gaussian distribution $\mathcal{N}(0, \sigma_a^2)$.

We aim to maximize $\mathcal{J}_K$ by conducting two processes for K iterations as follows. In iteration k: (1) We choose a node $i \in \mathcal{P}_k$ with the maximum expectation to get higher $\mathcal{J}_K$. It is based on all the learned $\boldsymbol{\theta}_a$, for $a = 1, 2, \dots, d$. (2) After querying the selected node i, we would get a reward $r_{i,k}$ from the oracle. Based on $r_{i,k}$ and $\mathbf{x}_i$, we update the $\boldsymbol{\theta}_a$, aiming to improve the node selection strategy, where a is the cluster that node i belongs to. We now introduce the details.

There are three types of information dominate node i, i.e., network structure, node attributes, and correlations to the B prototype nodes. We define $\mathbf{x}_i \in \mathbb{R}^{1\times(d+2)}$ as,

$$\mathbf{x}_i = [\mathbf{h}_i, \text{maxSimi}(i), \text{Corr}(i)], \tag{5.3}$$

where $\mathbf{h}_i$ is the attributed network embedding representation of node i, and maxSimi(i) denotes the maximum similarity from node i to all B prototype nodes. Corr(i) denotes the average of the correlations between node i and all prototype nodes. To make the algorithm efficient, in this chapter, $\mathbf{x}_i$ is fixed from the initialization. It is flexible to explore dynamic frameworks that update $\mathbf{x}_i$ in every iteration. Both the distance and correlation are based on the cosine similarities in the initial embedding representation space $\mathbf{H}_0$.

We now focus on each cluster a. Let $\mathbf{Y}_a \in \mathbb{R}^{y\times(d+2)}$ be a matrix that collects the information vectors of the y queried nodes in cluster a. Let $\mathbf{y}_a \in \mathbb{R}^{y\times 1}$ denote the corresponding y rewards. By leveraging ridge regression to estimate $\boldsymbol{\theta}_a$ and adding a penalty term λ to improve the estimation performance, we have,

$$\hat{\boldsymbol{\theta}}_a = (\mathbf{Y}_a^\top \mathbf{Y}_a + \lambda \mathbf{I})^{-1} \mathbf{Y}_a^\top \mathbf{y}_a. \tag{5.4}$$

Then for any $\delta > 0$, with probability at lease $1 - \delta$, the expectation of the reward of selecting a node i in cluster a is bounded by [102],

$$\mathbf{x}_i\hat{\boldsymbol{\theta}}_a - \alpha f_a(i) \leq \mathrm{E}[r_{i,k}|\mathbf{x}_i] \leq \mathbf{x}_i\hat{\boldsymbol{\theta}}_a + \alpha f_a(i), \tag{5.5}$$

where $\alpha = 1 + \sqrt{\ln(2/\delta)/2}$ is a constant, and

$$f_a(i) \triangleq \sqrt{\mathbf{x}_i(\mathbf{Y}_a^\top \mathbf{Y}_a + \lambda \mathbf{I})^{-1}\mathbf{x}_i^\top}. \tag{5.6}$$

The work of [2, 4] have shown that choosing the node maximizing the upper confidence bound $\mathbf{x}_i\hat{\boldsymbol{\theta}}_a + \alpha f_a(i)$ in each iteration achieves the best performance in maximizing $\mathcal{J}_K$. The key idea is, when a node with large $\mathbf{x}_i\hat{\boldsymbol{\theta}}_a$ is chosen, a high reward is expected, and such trial is an exploitation process. When a node with large $\alpha f_a(i)$ is chosen, this high variance means that we have limited knowledge on the model for cluster a, thus, we explore it more in the current trial. Jointly maximizing $\mathbf{x}_i\hat{\boldsymbol{\theta}}_a + \alpha f_a(i)$ makes a trade-off between exploitation and exploration. As a conclusion, in iteration k, the best node i for querying could be computed as follows.

$$\underset{i \in \mathcal{P}_k}{\text{maximize}} \quad g(i) = \mathbf{x}_i\hat{\boldsymbol{\theta}}_a + \alpha f_a(i). \tag{5.7}$$

5.3.4 Expert cognition quantization

We have defined the amount of expert cognition included in the k^{th} returned answer as $r_{i,k}$. We now introduce how to quantify it. The key idea is that incorporating more expert cognition could reduce the disagreements between the network proximity and node attribute proximity. Given a new node, experts usually jointly consider different types of information to get a comprehensive understanding, including its relation network and node attributes. It matches the observation that node attributes often highly tie in with the topological structure [67, 98]. However, in the data,

Algorithm 3: Network Embedding with Expert Cognition [37]

```
   Input: $\mathbf{G}$, $\mathbf{A}$, $d$, $B$, $K$, $\beta$, the oracle.
   Output: Cognition informed representation $\mathbf{H}$.
1  Select $B$ prototype nodes via k-medoids or random strategy;
2  Set the remaining nodes as $\mathcal{P}_1$ and split them into $d$ clusters;
3  Collect the information vectors of nodes in $\mathcal{P}_1$ based on Eq. (5.3);
4  Calculate $\mathbf{P}$ based on Eq. (5.8);
5  Calculate initial representation $\mathbf{H}_0$ based on an ANE framework;
6  Set all the $\mathbf{Q}_a \leftarrow \lambda\mathbf{I}$, $\mathbf{q}_a \leftarrow \mathbf{0}$, and calculate all the $g(i)$;
7  for $k = 1 : K$ do
8      Choose node the $i$ with maximum $g(i)$;
9      Query the oracle node $i$ and get the answer;
10     Incorporate node $i$'s cognition edge to $\mathbf{G}$;
11     Calculate $r_{i,k}$ based on Eq. (5.9);
12     Update $\mathbf{Q}_a \leftarrow \mathbf{Q}_a + \mathbf{x}_i^\top \mathbf{x}_i$ and $\mathbf{q}_a \leftarrow \mathbf{q}_a + r_{i,k}\mathbf{x}_i^\top$;
13     Remove node $i$ from the set $\mathcal{P}_k$ and get $\mathcal{P}_{k+1}$;
14     For nodes in cluster $a$, set $g(i) \leftarrow \mathbf{x}_i\mathbf{Q}_a^{-1}\mathbf{q}_a + \alpha\sqrt{\mathbf{x}_i\mathbf{Q}_a^{-1}\mathbf{x}_i^\top}$;
15 end
16 Calculate cognition informed $\mathbf{H}$ based on current $\mathbf{G}$ and $\mathbf{A}$.
```

there always exist some disagreements between the two sources, and we quantify them as follows.

$$\mathbf{P} = |\mathbf{S}^{(\mathbf{U}^{(G)})} - \mathbf{S}^{(\mathbf{U}^{(A)})}|, \tag{5.8}$$

where $|\cdot|$ calculates each absolute value. $\mathbf{U}^{(G)}$ and $\mathbf{U}^{(A)}$ are the embedding representations of the initial network and node attributes. We embed them to alleviate the effect of noise and missing data. Let j be the most similar prototype node of i based on the answer. Since adding a cognition edge (i, j) could alleviate the disagreement, we quantify the learned expert cognition in the k^{th} iteration as,

$$r_{i,k} = p_{ij}. \tag{5.9}$$

5.3.5 Network embedding with expert cognition

NEEC could be generalized to advance different embedding algorithms. We now introduce a simple ANE algorithm.

5.3.5.1 A simple attributed network embedding algorithm

The main idea of the spectrum technique is to enforce nodes i and j with larger proximity s_{ij} to be closer to each other in the embedding space. It can be achieved by minimizing the loss function $\mathcal{J}_{\text{spec}}$, i.e.,

$$\mathcal{J}_{\text{spec}} = \frac{1}{2}\sum\nolimits_{i,j=1}^{n} s_{ij}\|\frac{\mathbf{u}_i}{\sqrt{d_i}} - \frac{\mathbf{u}_j}{\sqrt{d_j}}\|_2^2, \tag{5.10}$$

where s_{ij} is the proximity in $\mathbf{S}^{(G)}$. d_i is the sum of the i^{th} row of $\mathbf{S}^{(G)}$, and it is employed for normalization. $\mathbf{u}_i$ needs to be orthonormal to avoid being arbitrary. We could further rewrite it as a maximization problem by defining $\mathcal{J}_G = 1 - \mathcal{J}_{\text{spec}}$, i.e.,

$$\begin{aligned} \underset{\mathbf{U}^{(G)}}{\text{maximize}} \quad & \mathcal{J}_G = \text{Tr}(\mathbf{U}^{(G)\top}\mathcal{L}^{(G)}\mathbf{U}^{(G)}) \\ \text{subject to} \quad & \mathbf{U}^{(G)\top}\mathbf{U}^{(G)} = \mathbf{I}, \end{aligned} \tag{5.11}$$

where $\mathcal{L}^{(G)} = \mathbf{D}^{(G)-\frac{1}{2}}\mathbf{S}^{(G)}\mathbf{D}^{(G)-\frac{1}{2}}$ is the graph Laplacian [101]. $\mathbf{D}^{(G)}$ is the degree matrix of $\mathbf{S}^{(G)}$, with d_i in the diagonal.

Similarly, we apply the same technique to the node attribute information $\mathbf{A}$, and calculate its embedding representation $\mathbf{U}^{(A)}$ by maximizing $\mathcal{J}_A = \text{Tr}(\mathbf{U}^{(A)\top}\mathcal{L}^{(A)}\mathbf{U}^{(A)})$, with $\mathbf{U}^{(A)\top}\mathbf{U}^{(A)} = \mathbf{I}$.

We apply a simple strategy to assimilate the network and node attributes towards a joint node proximity as follows.

$$\mathbf{S}^{(H)} = \beta\mathbf{S}^{(G)} + (1-\beta)\mathbf{S}^{(A)}, \tag{5.12}$$

where β balances the contributions of network and node attributes. Similarly, we could define the Laplacian $\mathcal{L}^{(H)} = \mathbf{D}^{(H)-\frac{1}{2}}\mathbf{S}^{(H)}\mathbf{D}^{(H)-\frac{1}{2}}$, where $\mathbf{D}^{(H)}$ is the degree matrix of $\mathbf{S}^{(H)}$, and apply the normalized spectral embedding technique to learn $\mathbf{H}$ as follows,

$$\begin{aligned} \underset{\mathbf{H}}{\text{maximize}} \quad & \mathcal{J} = \text{Tr}(\mathbf{H}^\top\mathcal{L}^{(H)}\mathbf{H}) \\ \text{subject to} \quad & \mathbf{H}^\top\mathbf{H} = \mathbf{I}. \end{aligned} \tag{5.13}$$

5.3.5.2 Embed the learned expert cognition

We summary the processes of NEEC in Algorithm 3. Given an attributed network with $\mathbf{G}$ and $\mathbf{A}$, we first select B prototype nodes and set the remaining nodes as the initial query pool $\mathcal{P}_1$. To choose the optimal K nodes from $\mathcal{P}_1$, we perform a contextual bandit algorithm based on Eq. (5.7). It has two items that are not necessary to be computed from scratch. We denote them as follows.

$$\mathbf{Q}_a = \mathbf{Y}_a^\top \mathbf{Y}_a + \lambda \mathbf{I}, \quad \text{and} \quad \mathbf{q}_a = \mathbf{Y}_a^\top \mathbf{y}_a. \tag{5.14}$$

Thus, in each iteration k, we update them according to the rules in line 12. Then we select the node with the maximum explanation upper confidence bound as defined in Eq. (5.7). After querying the oracle, we incorporate the answer into $\mathbf{G}$ as a cognition edge. When all the budget is used, we apply an attributed network embedding method to the current network $\mathbf{G}$ and $\mathbf{A}$, and jointly embed them towards a more informative cognition informed representation $\mathbf{H}$.

5.4 Experiments

We apply NEEC on three real-world attributed networks to demonstrate its effectiveness and generalizability. In the experiments, we target to investigate three questions. (1) How effective are the proposed prototype-based form of queries and the query selection algorithm? (2) Could the learned expert cognition be used to improve different embedding methods? (3) What are the impacts of the parameters B, K, and d, on the expert cognition learning?

5.4.1 Experimental settings

We use the same experimental settings as Chapter 2. The three datasets used in the experiments, including BlogCatalog, Flickr, and ACM, have been introduced in Chapters 2 and 3. If it is not specified, we set $d = 100$ and $B = 9$. All results are the means of ten test runs.

To create the oracle, for each dataset, we randomly sample a certain percentage of the entire edges and attributes as the initial attributed network. The remaining data is considered as the cognition of the oracle, so he/she answers the queries based on the entire original dataset. The

	BlogCatalog				Flickr				ACM			
Training	10%	25%	50%	100%	10%	25%	50%	100%	10%	25%	50%	100%
Node Num	1,455	2,078	3,118	5,196	2,121	3,030	4,545	7,575	4,616	6,594	9,890	1,6484
K	242	495	1,114	3,092	338	691	1,556	4,316	348	709	1,598	4,435
Initial	0.287	0.359	0.430	0.520	0.261	0.304	0.358	0.432	0.232	0.270	0.319	0.400
RandomPair	0.286	0.358	0.431	0.520	0.261	0.306	0.365	0.435	0.231	0.267	0.317	0.412
LinkPredict	0.287	0.360	0.430	0.523	0.261	0.303	0.359	0.433	0.230	0.278	0.310	0.392
AddKEdges	0.312	0.379	0.462	0.547	0.255	0.312	0.363	0.457	**0.258**	**0.300**	**0.380**	**0.508**
HiddenEdge	0.298	0.370	0.437	0.529	0.265	0.301	0.363	0.436	0.232	0.275	0.322	0.418
w/o_Bandit	0.295	0.366	0.447	0.556	0.259	0.326	0.368	0.439	0.238	0.265	0.306	0.391
NEEC_Rand	**0.317**	**0.393**	0.463	0.558	0.277	**0.367**	0.433	0.508	0.235	0.288	0.355	0.447
NEEC_Kmed	0.313	0.389	**0.469**	**0.571**	**0.281**	0.364	**0.443**	**0.515**	0.242	0.284	0.363	0.445
RandomPair	−0.47%	−0.27%	0.07%	−0.07%	0.04%	0.87%	1.87%	0.64%	−0.59%	−1.35%	−0.63%	3.11%
LinkPredict	−0.03%	0.35%	−0.07%	0.41%	0.00%	−0.36%	0.15%	0.15%	−0.80%	2.90%	−2.92%	−2.09%
AddKEdges	8.43%	5.55%	7.36%	5.14%	−2.28%	2.79%	1.20%	5.83%	**10.85%**	**10.96%**	**19.23%**	**27.00%**
HiddenEdge	3.61%	3.06%	1.52%	1.68%	1.60%	−0.80%	1.41%	0.81%	−0.26%	1.81%	0.93%	4.56%
w/o_Bandit	2.64%	2.06%	3.82%	6.75%	−0.80%	7.43%	2.61%	1.58%	2.30%	−2.07%	−4.13%	−2.17%
NEEC_Rand	**10.41%**	**9.63%**	7.69%	7.27%	6.24%	**20.87%**	20.78%	17.43%	1.07%	6.74%	11.25%	11.66%
NEEC_Kmed	9.03%	8.34%	**9.01%**	**9.67%**	**7.64%**	19.78%	**23.60%**	**19.22%**	4.26%	5.16%	13.71%	11.30%

Table 5.1: Classification performance of different methods in terms of micro-average. [37]

oracle determines the most similar prototype nodes based on the similarity of the original node attributes. Then the expert cognition learning methods could be employed to query the oracle and enhance the initial networks. A network embedding or attributed network learning method would be applied to both the initial network and the learned network. In such a way, we are able to evaluate the amount of learned expert cognition by measuring the performance improvement of the embedding representation.

5.4.2 Effectiveness of NEEC

To study the first question proposed at the beginning of this section, we compare NEEC with three types of baselines. First, to investigate the effectiveness of the proposed prototype-based form of queries, we include RandomPair, LinkPredict, and AddKEdges. Second, to demonstrate the effectiveness of the proposed way of formulating expert cognition, we include HiddenEdge. Third, to study the effectiveness of the proposed query selection algorithm, we include a variation of NEEC, i.e., w/o_Bandit. For NEEC, we include both k-medodis and random prototype node selection algorithms. The details of them are described as follows.

- *RandomPair*: No prototype node is defined. It randomly selects $B + K$ unconnected node pairs for querying. The oracle will indicate an edge, if the two nodes are linked in the original dataset or have an original node attribute similarity that is greater than a threshold.

- *LinkPredict*: Similar to RandomPair, but the $B + K$ pairs are selected from the unconnected pairs based on the probabilities of having edges, which are based on the weighted-ℓ_1 edge features [24] in spectral embedding space.

- *HiddenEdge*: It queries the oracle based on prototype nodes, but without using the cognition edges. It adds edges between the selected node i and any prototype nodes that have latent relationships with i or are highly similar to i.

- *w/o_Bandit*: NEEC without using the proposed query selection algorithm. It chooses the top K important nodes with the maximum eigenvector centrality [71].

- *AddKEdges*: It randomly queries the oracle a large number of node pairs until K edges were added. This baseline unfairly takes much more human effort than all others.

5.4.2.1 Form of queries investigation

To study the effectiveness of querying with prototype nodes, we compare NEEC with RandomPair, LinkPredict, and AddKEdges. All these three baselines query the oracle without a prototype. For RandomPair and LinkPredict, the total number of queries is set as $B + K$, such that the oracle needs to check at least $B + K$ nodes and pay more effort than in NEEC. AddKEdges needs to make an extremely large number of queries. The classification performance w.r.t. micro-average is shown in Table 5.1. All of them use the same embedding algorithm in Section 5.3.5. From the results, we observe that NEEC outperforms all the baselines on all the three datasets. For instance, on Flickr, both RandomPair and LinkPredict have almost no improvement, while NEEC_Kmed achieves 19.22% improvement comparing with the embedding of the initial attributed network. Even though AddKEdges performs a large number of queries, NEEC improves more on BlogCatalog and Flickr. On ACM, NEEC achieves less improvement. The reason is that AddKEdges needs to perform up to 9.4×10^6 queries.

To further investigate the performance of NEEC under different training percentages, i.e., the percentage of data in training group that has been used, we vary it from 10% to 100%. The results in Table 5.1 show that NEEC consistently achieves higher performance than all baselines except AddKEdges. For instance, when the training percentage is 25% on Flickr, NEEC_Rand achieves 20.87% improvement. It also demonstrates that the prototype-based form of queries is effective in learning expert cognition.

5.4.2.2 *Effectiveness of cognition edge investigation*

To study the effectiveness of the proposed way of transforming expert cognition into concrete data, we compare NEEC with HiddenEdge, which models the expert cognition as latent edges. Their performance in items of different training percentage is summarized in Table 5.1. As we can see, HiddenEdge achieves limited improvement. For instance, NEEC_Kmed achieves 9.67% while HiddenEdge achieves 1.68%, when the training percentage is 100% on BlogCatalog. It demonstrates the effectiveness of the proposed cognition edges.

5.4.2.3 *Prototype node and query selection algorithms investigation*

We provide two algorithms to select the prototype nodes, i.e., NEEC_Rand and NEEC_Kmed. As shown in Table 5.1, NEEC_Rand might achieve slightly better performance when the training percentage is 10% or 25%. It is because when the number of nodes is small, the initial information would be too limited to perform the clustering well in NEEC_Kmed.

To study the effectiveness of our contextual bandit algorithm in query selection, we compare NEEC with w/o_Bandit. The performance with respect to different training percentage is shown in Table 5.1. We observe that NEEC achieves more improvement. For example, on Flickr, NEEC_Rand achieves 17.43% improvement while w/o_Bandit achieves 1.58% improvement.

5.4.2.4 *Effectiveness and efficiency of NEEC investigation*

Up till now, we have demonstrated the effectiveness of each component of NEEC. The number of queries we performed also demonstrates the effectiveness of NEEC. We summarize the number of queries K and performance improvement in Table 5.1. As we can see, NEEC achieves significant improvement with a small number of queries. For example, NEEC_Kmed achieves 19.78%

Dataset	BlogCatalog	Flickr	ACM
Average Time (s)	$2.97e{-}003$	$8.02e{-}003$	$2.28e{-}002$

Table 5.2: The computation time of each iteration in NEEC. [37]

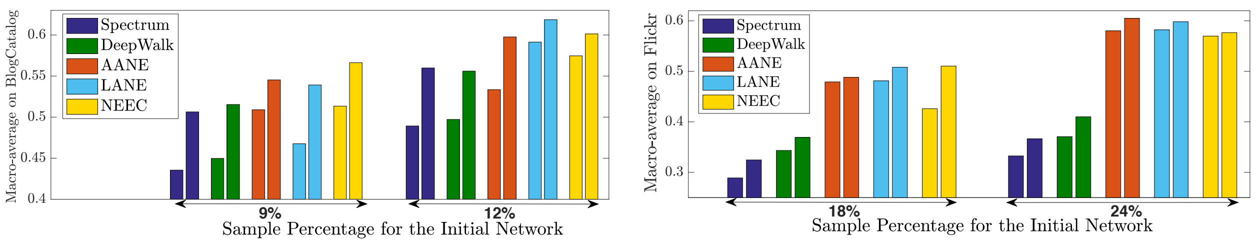


Figure 5.2: The classification performance of five embedding methods before and after incorporating the expert cognition. [37]

improvement by conducting 691 queries in a network with $3{,}030$ nodes on Flickr.

NEEC is also an efficient interactive system that could respond the oracle immediately. The average computation time of each iteration in NEEC is shown in Table 5.2. For example, on Flickr, NEEC needs 8.02 milliseconds to update the system and compute a new query. It verifies the efficiency of NEEC.

5.4.3 Generalizability evaluation

We now study how general is the expert cognition learning framework NEEC. We include two network embedding methods, i.e., Spectrum and DeepWalk, and two ANE methods, i.e., AANE and LANE, as baselines. We have introduced DeepWalk and AANE in Chapter 2.

- *Spectrum* [101]: It performs spectral embedding on the pure topological structure to evaluate the amount of information that we have learned in the network space.
- *LANE* [35]: It is used for incorporating labels into network embedding. We set the labels as zeros. It jointly embeds the network and node attributes by maximizing their correlations based on the spectral embedding.

The performance of the five embedding methods before and after incorporating the expert cognition is shown in Figure 5.2. We omit the result on ACM since we obtain similar observations.

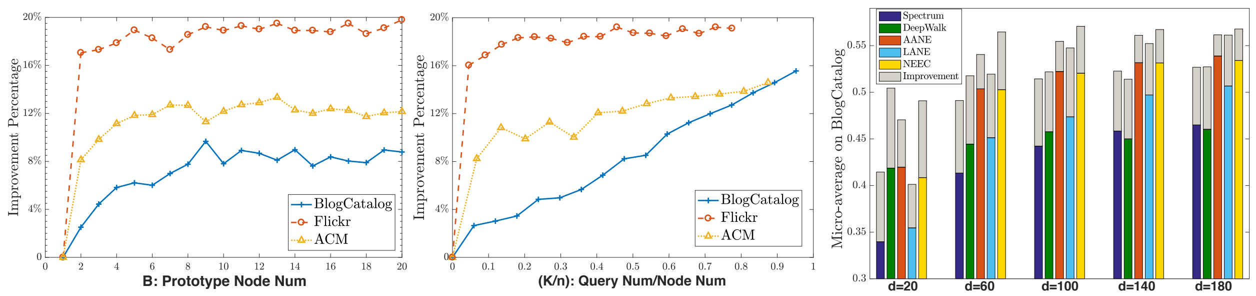


Figure 5.3: Performance improvement of embedding methods after incorporating the expert cognition with parameters varying. [37]

From the figure, we find that all the five embedding methods have significant performance improvements after incorporating the expert cognition. For example, DeepWalk achieves 14.60% improvement and LANE achieves 15.29% improvement on BlogCatalog. We further increase one-third of the sample percentage for the initial network to study its impact. From the results in Figure 5.2, the same observations are observed, i.e., all the five methods have significant performance improvements after incorporating the expert cognition. We also find that the amount of improvement decreased. This can be explained by the fact that, as the sample percentage increases, the amount of knowledge that the oracle obtained decreases.

5.4.4 Parameter analysis

We now study the impacts of the number of prototype nodes B, the number of queries K, and the embedding representation dimension d. The performance improvement w.r.t. B and $\frac{K}{n}$ after incorporating the expert cognition is shown in Figure 5.3. As we can see, the performance improvement increases as B increases on all the three datasets. It is because more prototypes tend to create more accurate cognition of nodes. But it also takes more human effort. The performance improvement increases rapidly at the beginning and then increases smoothly thereafter. Similar observations are made for the parameter K. Larger query number means more expert cognition, but their correlation is not always linear. We also vary d to see the performance improvement of the five embedding methods and show the results in Figure 5.3. The gray bars denote the corresponding performance improvement. We see that all methods perform better after incorporating the cognition edges as d increases from 20 to 180. This also verifies the generalizability of NEEC.

6. CONCLUSION AND FUTURE WORK

6.1 Conclusion

In this thesis, we propose a series of embedding and interactive frameworks to handle the heterogeneous information learning in real-world networks at scale. Attributed network embedding aims to learn a low-dimensional continuous vector representation for each node, such that information in the network and node attributes could be preserved. Challenges include the heterogeneity of networks and node attributes. Also, real-world networks often have a large number of nodes and a high-dimensional node attributes. We systematically develop three attributed network embedding algorithms, including AANE, FeatWalk, and GraphRNA.

First, AANE could effectively model attributed networks by incorporating its node proximity into network embedding in a distributed manner. It learns a low-dimensional representation based on the decomposition of the node attributes affinity matrix and the embedding difference between connected nodes. A distributed optimization algorithm is developed to decompose the complex problem into many sub-problems of low complexity, which could be solved by sub-workers in parallel. A variation of AANE to handle streaming networks is also proposed. Experiments on the three real-world attributed networks demonstrate the effectiveness and efficiency of AANE. Second, FeatWalk could encode the network and node attributes into a unified latent representation. Without calculating any similarity measure among nodes' attributes, we design an alternative way to simulate the similarity-based random walks among nodes, which samples the local node similarities and preserves them in walking trajectories. Along with the trajectories learned via random walks on the network relations, we apply a scalable word embedding algorithm to learn the joint representations of nodes from these trajectories. Experiments on the four real-word datasets validate the scalability and effectiveness of FeatWalk. Third, we propose an elegant walking mechanism - AttriWalk, which could conduct collaboratively sampling within the network and node attributes. Motivating from the recent advances of deep learning techniques in representation learning, we further design a tailored graph neural network architecture upon AttriWalk, named GraphRNA, to conduct attributed network embedding. GraphRNA converts the complex attributed node interactions into a series of informative node index sequences based on AttriWalk, and encodes them into unified vector representations via graph recurrent networks. Evaluation on real-world datasets demonstrates the effectiveness of GraphRNA comparing with diverse baselines.

AANE, FeatWalk, and GraphRNA have different theoretical foundations, including distributed matrix factorization, spectral embedding, random walks, and graph neural networks. Thus, they have different properties and could be employed to handle different real-world scenarios.

Expert cognition is an essential type of knowledge that could advance various real-world data analysis tasks, including attributed network embedding. Learning and modeling expert cognition is promising but challenging as it is often abstract, diverse, and laborious to obtain. Thus, we explore an interactive framework - NEEC to involve data scientists and domain experts to help in learning more informative latent representations. Instead of directly modeling expert cognition, we learn it from the oracle by performing a number of concise but effective queries. The queries are carefully designed based on the exemplar theory and prototype theory, and systematically selected via a contextual bandit algorithm. Based on the returned answers from the oracle, the meaningful but abstract expect cognition is modeled as new cognition edges, which could be directly added into the network. Thus, the learned expert cognition could be incorporated into the latent representation by any network embedding algorithm. Experiments on real-world datasets demonstrate the effectiveness and generalizability of NEEC.

ANE is crucial to the field since their learned node representations could well preserve all the heterogeneous information in the systems, and directly be employed as node features to perform various analysis applications. This thesis will broadly impact fields such as Information Retrieval, Social Computing, and Health Informatics.

6.2 Future Research Directions

While performing research on learning from attributed networks, I have accumulated a number of rewarding but challenging research questions. They could be categorized into two direc-

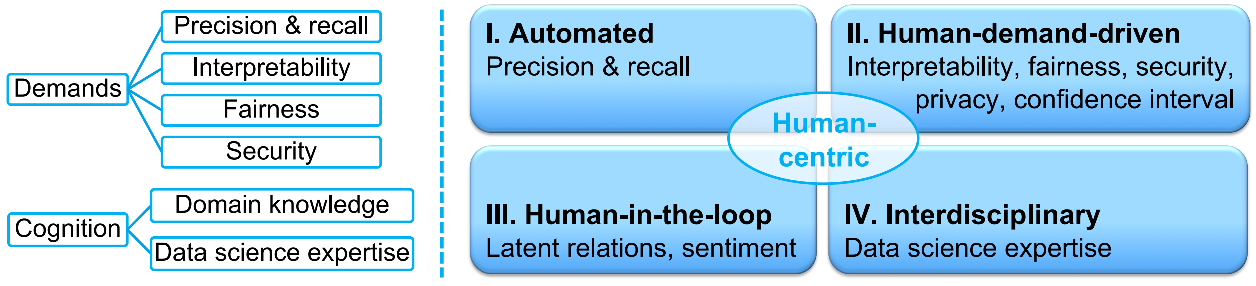


Figure 6.1: Human-centric network learning based on the demands and cognition of humans.

tions, i.e., human-centric network learning and tailored network learning for high-impact interdisciplinary studies, as illustrated in Figure 6.1.

6.2.1 Human-centric network learning

Given that most network analysis algorithms are data-driven, it is essential to explore human-centric network learning. We roughly categorize the research topics into three classes.

6.2.1.1 Automated network learning

We first focus on humans' demands on precision and recall. There are many available network embedding algorithms and end-to-end learning methods, including graph neural networks with various architectures. They have different properties, with performance vary in different networks and tasks. For example, when attention heads in graph attention networks [100] are selected carefully, they could achieve prominent learning performances on citation networks and protein-protein interaction networks. GraphSAGE [26] has been shown to be sensitive to the dimension of hidden units. These hand-crafted architectures not only require extensive search in the design space through many trials, but also tend to obtain suboptimal performance when they are transferred to other graph-structured data. Thus, it is desired to have a system that can automatically identify and tune an appropriate network learning algorithm or graph neural architecture, for a given network and task. Such system would enable the analysis of real-world networks more easily accessible to data consumers who have a limited data science background and data scientists who want to identify suitable models quickly.

6.2.1.2 Human-demand-driven network learning

Human demands can go beyond precision and recall. As illustrated by Figure 6.1, we also care about transparency, interpretability, fairness, privacy, security, and confidence intervals.

For instance, domain experts would trust and effectively utilize prediction results, only if they could apprehend the models or results. Similar to many other machine learning methods, network embedding results remain hard to be understood by users. Each dimension in the embedding space usually does not have any specific meaning, thus it is difficult to comprehend how the embedding instances are distributed in the reconstructed space. In addition, there are two information sources in attributed networks, it is challenging to specify which source has been actively used in generating the embedding results. Thus, we plan to investigate the interpretation of network embedding, aiming to understand how instances are distributed in embedding space, as well as explore the factors that lead to the embedding results.

Also, to enhance the security in network learning, I plan to explore the adversarial network learning. Deep network learning, such as graph neural networks, has achieved superior empirical performance. But deep models could be easily attacked. Given that there are many security-related network analysis applications, e.g., financial fraud detection and fake news detection, it is urgent to investigate adversarial network learning to enhance the robustness of graph neural networks.

Another example is network learning with confidence intervals, which is essential to the decision making on networks. It is a nontrivial task to learn confidence intervals on networks, especially for graph neural networks.

Thus, network learning with interpretability, fairness, privacy, and security are all promising, essential, and in the early stages.

6.2.1.3 Human-in-the-loop network learning

We now focus on the cognition of humans, as illustrated in Figure 6.1. Developing interactive models to learn from domain experts and data scientists is also a promising way of advancing network analysis. As introduced in Chapter 5, experts could know extra prior knowledge of data, such as the understanding of domain knowledge and the perception of latent relations. The proposed framework NEEC in Chapter 5 explores an effective and general way to learn and embed the cognition of latent node similarities. There are many other types of domain knowledge such as the comprehension of semantic information and the awareness of conventions. It is challenging but rewarding to model and incorporate more types of expert cognition.

6.2.2 Interdisciplinary research

There are many opportunities in leveraging advances in network learning to address real-world problems in other disciplines and promote social good. I elaborate on three exemplary fields.

First, *social data science*. Tons of user-generated content are available on various online social platforms. These platforms often have different foci and have become an essential source for studying the 21st century society. Scalable network learning serves as an effective tool to enable efficient studies on these large-scale social networks. For example, network embedding could accelerate the moral behavior study. Recent sociological studies show that seven moral rules are universal, including family values, group loyalty, reciprocity, bravery, respecting others, fairness, and property rights. Understanding moral behavior helps us gain insights into human sociality. E.g., many social conflicts are essentially conflicts of ordering moral rules. When studying the evolution of moral behavior, network learning tools could be used to discover patterns in social dilemma games. For example, when studying network reciprocity, social roles such as friends and strangers could be described by node attributes. The embedding representations of communities could be used to characterize cooperative groups. Other examples include the early warning of cyberbullying, detection of fake news, as well as demographic inference and representative population estimation.

Second, *personalized online education*. It is a long-standing dream to educate individual students according to their aptitudes, interests, and learning styles. For example, some people learn faster via finding answers to questions, while some respond more effectively to associative learning. In online education, students, assignments, multimedia content materials, forum posts, interact with each other and form a large network. Network learning techniques could be applied to perform learning path recommendation. Fertile node attributes such as the descriptions of assignments, content in materials, attributes and posts of students, are also available. Thus, attributed network embedding could serve as an efficient analysis tool. Tailored interactive models could be explored to accelerate the recommendation.

Third, *drug-drug interaction prediction*. The pharmacologic effect of a drug might be altered by another drug or special food, when they are administered simultaneously. Such drug-drug and drug-food interactions could be modeled as links. By considering drug descriptions has node attributes, attributed network embedding could be applied to predict interactions between a new drug and existing drugs.

REFERENCES

[1] E. Achtert, H.-P. Kriegel, E. Schubert, and A. Zimek. Interactive data mining with 3d-parallel-coordinate-trees. In *ACM Special Interest Group on Management of Data*, pages 1009–1012, 2013.

[2] R. Agrawal. Sample mean based index policies by o(log n) regret for the multi-armed bandit problem. *Advances in Applied Probability*, 27(4):1054–1078, 1995.

[3] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola. Distributed large-scale natural graph factorization. In *International World Wide Web Conference*, pages 37–48, 2013.

[4] P. Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3:397–422, 2002.

[5] L. Backstrom and J. Leskovec. Supervised random walks: Predicting and recommending links in social networks. In *ACM International Conference on Web Search and Data Mining*, pages 635–644, 2011.

[6] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2015.

[7] Y. Baram, R. El-Yaniv, and K. Luz. Online choice of active learning algorithms. *Journal of Machine Learning Research*, 5:255–291, 2004.

[8] D. Bouneffouf, R. Laroche, T. Urvoy, R. Féraud, and R. Allesiardo. Contextual bandit for active learning: Active thompson sampling. In *International Conference on Neural Information Processing*, pages 405–412, 2014.

[9] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[10] S. Chang, W. Han, J. Tang, G.-J. Qi, C. C. Aggarwal, and T. S. Huang. Heterogeneous network embedding via deep architectures. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 119–128, 2015.

[11] J. Chen, T. Ma, and C. Xiao. FastGCN: Fast learning with graph convolutional networks via importance sampling. In *International Conference on Learning Representations*, 2018.

[12] E. C. Chi and K. Lange. Splitting methods for convex clustering. *Journal of Computational and Graphical Statistics*, 24(4):994–1013, 2015.

[13] P. Cui, X. Wang, J. Pei, and W. Zhu. A survey on network embedding. *IEEE Transactions on Knowledge and Data Engineering*, 2017.

[14] R. Davies-Jones. A review of supercell and tornado dynamics. *Atmospheric Research*, pages 274–291, 2015.

[15] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Conference on Neural Information Processing Systems*, pages 3844–3852, 2016.

[16] G. DeJong and S. H. Lim. Explanation-based learning. *Encyclopedia of Machine Learning*, pages 388–392, 2011.

[17] L. Devroye. Sample-based non-uniform random variate generation. In *Winter Simulation Conference*, pages 260–265, 1986.

[18] X. Ding, B. Liu, and P. S. Yu. A holistic lexicon-based approach to opinion mining. In *ACM International Conference on Web Search and Data Mining*, pages 231–240, 2008.

[19] Y. Dong, N. V. Chawla, and A. Swami. metapath2vec: Scalable representation learning for heterogeneous networks. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 135–144, 2017.

[20] A. El Alaoui, X. Cheng, A. Ramdas, M. J. Wainwright, and M. I. Jordan. Asymptotic behavior of ℓ_p-based Laplacian regularization in semi-supervised learning. *Journal of Machine Learning Research*, 49:1–28, 2016.

[21] R. Eyal, A. Rosenfeld, S. Sina, and S. Kraus. Predicting and identifying missing node information in social networks. *ACM Transactions On Knowledge Discovery From Data*, 8(3):1–35, 2014.

[22] H. Gao and H. Huang. Deep attributed network embedding. In *International Joint Conference on Artificial Intelligence*, pages 3364–3370, 2018.

[23] X. Gao and S.-Z. Zhang. First-order algorithms for convex optimization with nonseparable objective and coupled constraints. *Journal of the Operations Research Society of China*, 5(2):131–159, 2017.

[24] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 855–864, 2016.

[25] D. Hallac, J. Leskovec, and S. Boyd. Network lasso: Clustering and optimization in large graphs. In *KDD*, pages 387–396, 2015.

[26] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Conference on Neural Information Processing Systems*, pages 1024–1034, 2017.

[27] M. R. Hestenes. Multiplier and gradient methods. *Journal of Optimization Theory and Applications*, 4(5):303–320, 1969.

[28] T. D. Hocking, A. Joulin, F. Bach, and J.-P. Vert. Clusterpath: An algorithm for clustering using convex fusion penalties. *International Conference on Machine Learning*, pages 745–752, 2011.

[29] H. Hoefling. A path algorithm for the fused lasso signal approximator. *Journal of Computational and Graphical Statistics*, 19(4):984–1006, 2010.

[30] S. C. H. Hoi, R. Jin, and M. R. Lyu. Large-scale text categorization by batch mode active learning. In *International World Wide Web Conference*, pages 633–642, 2006.

[31] A. Holzinger. Interactive machine learning for health informatics: When do we need the human-in-the-loop? *Brain Informatics*, 3(2):119–131, 2016.

[32] X. Hu, J. Tang, H. Gao, and H. Liu. Unsupervised sentiment analysis with emotional signals. In *International World Wide Web Conference*, pages 607–618, 2013.

[33] X. Hu, L. Tang, J. Tang, and H. Liu. Exploiting social relations for sentiment analysis in microblogging. In *ACM International Conference on Web Search and Data Mining*, pages 537–546, 2013.

[34] J. Huang, F. Nie, H. Huang, Y.-C. Tu, and Y. Lei. Social trust prediction using heterogeneous networks. *ACM Transactions On Knowledge Discovery From Data*, 7(4):1–21, 2013.

[35] X. Huang, J. Li, and X. Hu. Label informed attributed network embedding. In *ACM International Conference on Web Search and Data Mining*, pages 731–739. doi.org/10.1145/3018661.3018667, 2017.

[36] X. Huang, J. Li, N. Zou, and X. Hu. A general embedding framework for heterogeneous information learning in large-scale networks. *ACM Transactions on Knowledge Discovery from Data*, 12(6), 2018.

[37] X. Huang, Q. Song, J. Li, and X. Hu. Exploring expert cognition for attributed network embedding. In *ACM International Conference on Web Search and Data Mining*, pages 270–278. doi.org/10.1145/3159652.3159655, 2018.

[38] X. Huang, Q. Song, Y. Li, and X. Hu. Graph recurrent networks with attributed random walks. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 732–740. doi.org/10.1145/3292500.3330941, 2019.

[39] X. Huang, Q. Song, F. Yang, and X. Hu. Large-scale heterogeneous feature embedding. In *AAAI Conference on Artificial Intelligence*, volume 33, pages 3878–3885. doi.org/10.1609/aaai.v33i01.33013878, 2019.

[40] Z. Huo, X. Huang, and X. Hu. Link prediction with personalized social influence. In *AAAI Conference on Artificial Intelligence*, 2018.

[41] L. Jian, J. Li, K. Shu, and H. Liu. Multi-label informed feature selection. In *International Joint Conference on Artificial Intelligence*, pages 1627–1633, 2016.

[42] Y. Jin and J. F. JaJa. Learning graph-level representations with gated recurrent neural networks. *arXiv preprint arXiv:1805.07683*, 2018.

[43] I. T. Jolliffe. Principal component analysis and factor analysis. In *Principal Component Analysis*, pages 115–128. Springer, 1986.

[44] W. Karwowski. *International Encyclopedia of Ergonomics and Human Factors*. CRC Press, 2001.

[45] T. N. Kipf and M. Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[46] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

[47] D. Kuang, C. Ding, and H. Park. Symmetric nonnegative matrix factorization for graph clustering. In *SIAM International Conference on Data Mining*, pages 106–117, 2012.

[48] A. Kumar, P. Rai, and H. Daume. Co-regularized multi-view spectral clustering. In *Conference on Neural Information Processing Systems*, pages 1413–1421, 2011.

[49] T. M. V. Le and H. W. Lauw. Probabilistic latent document network embedding. In *IEEE International Conference on Data Mining*, pages 270–279, 2014.

[50] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(436–444), 2015.

[51] J. Lee and S. Prabhakar. A3embed: Attribute association aware network embedding. In *International World Wide Web Conference*, pages 1243–1251, 2018.

[52] D. D. Lewis and W. A. Gale. A sequential algorithm for training text classifiers. In *Special Interest Group on Information Retrieval*, pages 3–12, 1994.

[53] J. Li, H. Dani, X. Hu, J. Tang, Y. Chang, and H. Liu. Attributed network embedding for learning in a dynamic environment. In *ACM International Conference on Information and Knowledge Management*, 2017.

[54] J. Li, X. Hu, J. Tang, and H. Liu. Unsupervised streaming feature selection in social media. In *ACM International Conference on Information and Knowledge Management*, pages 1041–1050, 2015.

[55] J. Li, X. Hu, L. Wu, and H. Liu. Robust unsupervised feature selection on networked data. In *SIAM International Conference on Data Mining*, pages 387–395, 2016.

[56] J.-H. Li, C.-D. Wang, L. Huang, D. Huang, J.-H. Lai, and P. Chen. Attributed network embedding with micro-meso structure. In *Database Systems for Advanced Applications*, pages 20–36, 2018.

[57] L. Li, W. Chu, J. Langford, and R. E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *International World Wide Web Conference*, pages 661–670, 2010.

[58] Q. Li, Z. Han, and X.-M. Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI Conference on Artificial Intelligence*, pages 3538–3545, 2018.

[59] J. Liang, P. Jacobs, J. Sun, and S. Parthasarathy. Semi-supervised embedding in attributed networks with outliers. In *SIAM International Conference on Data Mining*, pages 153–161, 2018.

[60] L. Liao, X. He, H. Zhang, and T.-S. Chua. Attributed social network embedding. *IEEE Transactions on Knowledge and Data Engineering*, 30(12):2257–2270, 2018.

[61] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu. Learning entity and relation embeddings for knowledge graph completion. In *AAAI Conference on Artificial Intelligence*, pages 2181–2187, 2015.

[62] F. Lindsten, H. Ohlsson, and L. Ljung. Just relax and come clustering!: A convexification of k-means clustering. *Linköping University Electronic Press*, 2011.

[63] N. Liu, X. Huang, and X. Hu. Accelerated local anomaly detection via resolving attributed networks. In *International Joint Conference on Artificial Intelligence*, pages 2337–2343, 2017.

[64] P. V. Marsden. Homogeneity in confiding relations. *Social Networks*, 10(1):57–76, 1988.

[65] J. McAuley, R. Pandey, and J. Leskovec. Inferring networks of substitutable and complementary products. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, 2015.

[66] A. McCallum and K. Nigam. Employing em and pool-based active learning for text classification. In *International Conference on Machine Learning*, pages 350–358, 1998.

[67] M. McPherson, L. Smith-Lovin, and J. M. Cook. Birds of a feather: Homophily in social networks. *Annual Review of Sociology*, 27(1):415–444, 2001.

[68] M. Meila and J. Shi. A random walks view of spectral segmentation. 2001.

[69] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Conference on Neural Information Processing Systems*, pages 3111–3119, 2013.

[70] H. Narayanan, M. Belkin, and P. Niyogi. On the relation between low density separation, spectral clustering and graph cuts. In *Conference on Neural Information Processing Systems*, pages 1025–1032, 2006.

[71] M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, 2003.

[72] R. M. Nosofsky. Attention, similarity, and the identification-categorization relationship. *Journal of Experimental Psychology: General*, 115(1):39–57, 1986.

[73] M. Ou, P. Cui, J. Pei, and W. Zhu. Asymmetric transitivity preserving graph embedding. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1105–1114, 2016.

[74] S. Pan, J. Wu, X. Zhu, C. Zhang, and Y. Wang. Tri-party deep network representation. In *International Joint Conference on Artificial Intelligence*, pages 1895–1901, 2016.

[75] S. Pandey, D. Chakrabarti, and D. Agarwal. Multi-armed bandit problems with dependent arms. In *International Conference on Machine Learning*, 2007.

[76] H.-S. Park and C.-H. Jun. A simple and fast algorithm for k-medoids clustering. *Expert Systems with Applications*, 36(2):3336–3341, 2009.

[77] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, et al. Scikit-Learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[78] K. Pelckmans, J. De Brabanter, J. Suykens, and B. De Moor. Convex clustering shrinkage. In *PASCAL Workshop on Statistics and Optimization of Clustering Workshop*, 2005.

[79] J. Pennington, R. Socher, and C. Manning. GloVe: Global vectors for word representation. In *Empirical Methods in Natural Language Processing*, pages 1532–1543, 2014.

[80] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 701–710, 2014.

[81] G.-J. Qi, C. Aggarwal, Q. Tian, H. Ji, and T. S. Huang. Exploring context and content links in social media: A latent space method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(5):850–862, 2012.

[82] F. Radlinski, R. Kleinberg, and T. Joachims. Learning diverse rankings with multi-armed bandits. In *International Conference on Machine Learning*, pages 784–791, 2008.

[83] E. Rosch, C. B. Mervis, W. D. Gray, D. M. Johnson, and P. Boyes-Braem. Basic objects in natural categories. *Cognitive Psychology*, 8(3):382–439, 1976.

[84] B. Settles. Active learning literature survey. CS Technical Reports, University of Wisconsin–Madison, 2009.

[85] B. Settles and M. Craven. An analysis of active learning strategies for sequence labeling tasks. In *Empirical Methods in Natural Language Processing*, pages 1070–1079, 2008.

[86] A. N. Smith, E. Fischer, and C. Yongjian. How does brand-related user-generated content differ across youtube, facebook, and twitter? *Journal of Interactive Marketing*, 26(2):102–113, 2012.

[87] M. Taboada, J. Brooke, M. Tofiloski, K. Voll, and M. Stede. Lexicon-based methods for sentiment analysis. *Computational Linguistics*, 37(2):267–307, 2011.

[88] A. Taheri, K. Gimpel, and T. Berger-Wolf. Learning graph representations with recurrent neural network autoencoders. In *Proc. KDD Deep Learn. Day*, 2018.

[89] J. Tang, C. Aggarwal, and H. Liu. Node classification in signed social networks. In *SIAM International Conference on Data Mining*, pages 54–62, 2016.

[90] J. Tang, H. Gao, X. Hu, and H. Liu. Exploiting homophily effect for trust prediction. In *ACM International Conference on Web Search and Data Mining*, pages 53–62, 2013.

[91] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. LINE: Large-scale information network embedding. In *International World Wide Web Conference*, pages 1067–1077, 2015.

[92] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su. Arnetminer: Extraction and mining of academic social networks. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 990–998, 2008.

[93] L. Tang and H. Liu. Scalable learning of collective behavior based on sparse social dimensions. In *ACM International Conference on Information and Knowledge Management*, pages 1107–1116, 2009.

[94] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B*, pages 267–288, 1996.

[95] R. Tibshirani, M. Saunders, S. Rosset, J. Zhu, and K. Knight. Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society: Series B*, 67(1):91–108, 2005.

[96] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *Journal of Machine Learning Research*, 2:45–66, 2001.

[97] K. Tsuda, H. Shin, and B. Schölkopf. Fast protein classification with multiple networks. *Bioinformatics*, 21(suppl_2):ii59–ii65, 2005.

[98] O. Tsur and A. Rappoport. What's in a hashtag? content based prediction of the spread of ideas in microblogging communities. In *ACM International Conference on Web Search and Data Mining*, pages 643–652, 2012.

[99] K. Tu, P. Cui, X. Wang, F. Wang, and W. Zhu. Structural deep embedding for hyper-networks. In *AAAI Conference on Artificial Intelligence*, 2018.

[100] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[101] U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[102] T. J. Walsh, I. Szita, C. Diuk, and M. L. Littman. Exploring compact reinforcement-learning representations with linear regression. In *Uncertainty in Artificial Intelligence*, pages 591–598, 2009.

[103] D. Wang, P. Cui, and W. Zhu. Structural deep network embedding. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1225–1234, 2016.

[104] H. Wang, Q. Wu, and H. Wang. Learning hidden features for contextual bandits. In *ACM International Conference on Information and Knowledge Management*, pages 1633–1642, 2016.

[105] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang. Community preserving network embedding. In *AAAI Conference on Artificial Intelligence*, pages 203–209, 2017.

[106] A. B. Wilcox and G. Hripcsak. The role of domain knowledge in automating medical text report classification. *Journal of the American Medical Informatics Association*, 10(4):330–338, 2003.

[107] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Y. Chang. Network representation learning with rich text information. In *International Joint Conference on Artificial Intelligence*, pages 2111–2117, 2015.

[108] C. Yang, M. Sun, Z. Liu, and C. Tu. Fast network embedding enhancement via high order proximity approximation. In *International Joint Conference on Artificial Intelligence*, pages 19–25, 2017.

[109] H. Yang, S. Pan, P. Zhang, L. Chen, D. Lian, and C. Zhang. Binarized attributed network embedding. In *IEEE International Conference on Data Mining*, 2018.

[110] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *KDD*, pages 974–983, 2018.

[111] C. Zeng, Q. Wang, S. Mokhtari, and T. Li. Online context-aware recommendation with time varying multi-armed bandit. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2025–2034, 2016.

[112] J. Zhang, J. Tang, J. Li, Y. Liu, and C. Xing. Who influenced you? predicting retweet via social influence locality. *ACM Transactions On Knowledge Discovery From Data*, 9(3):1–26, 2015.

[113] L. Zhang, Q. Zhang, L. Zhang, D. Tao, X. Huang, and B. Du. Ensemble manifold regularized sparse low-rank approximation for multiview feature embedding. *Pattern Recognition*, 48(10):3102–3112, 2015.

[114] Y. Zhang. Incorporating phrase-level sentiment analysis on textual reviews for personalized recommendation. In *ACM International Conference on Web Search and Data Mining*, pages 435–440, 2015.

[115] Z. Zhang, H. Yang, J. Bu, S. Zhou, P. Yu, J. Zhang, M. Ester, and C. Wang. Anrl: Attributed network representation learning via deep neural networks. In *International Joint Conference on Artificial Intelligence*, pages 3155–3161, 2018.

[116] S. Zhu, K. Yu, Y. Chi, and Y. Gong. Combining content and link for classification using matrix factorization. In *Special Interest Group on Information Retrieval*, pages 487–494, 2007.

APPENDIX A

PROOF OF THEOREMS AND COROLLARIES

We now prove Theorem 2.3.

Proof. Our goal is to prove that $f(\mathbf{x}^{k+1}) \leq f(\mathbf{x}^k)$, when $\mathbf{x}^{k+1}$ and $\mathbf{x}^k$ have a relation as follows.

$$\mathbf{x}^{k+1} = (\textstyle\sum_j a_j b_j \mathbf{z}_j - \mathbf{c}) \times (2\mathbf{Q} + \textstyle\sum_j a_j b_j \mathbf{I})^{\dagger}.$$

First, for $\mathbf{x}^k \neq \mathbf{z}_j$, we define a new quadratic function of $\mathbf{x}$ as follows,

$$g(\mathbf{x}) \triangleq \mathbf{x}(\mathbf{Q} + 0.5\textstyle\sum_j a_j b_j \mathbf{I})\mathbf{x}^\top - (\textstyle\sum_j a_j b_j \mathbf{z}_j - \mathbf{c})\mathbf{x}^\top. \tag{A.1}$$

It is easy to prove that $g(\mathbf{x})$ achieves the global minimum at $\mathbf{x}^{k+1}$. Thus, for any vector $\mathbf{x}^k \neq \mathbf{z}_j$, we have $g(\mathbf{x}^{k+1}) \leq g(\mathbf{x}^k)$. By substituting $\mathbf{x}^{k+1}$ and $\mathbf{x}^k$ into $g(\mathbf{x})$, we get an inequation as follows.

$$\mathbf{x}^{k+1}\mathbf{Q}{\mathbf{x}^{k+1}}^\top + \mathbf{c}{\mathbf{x}^{k+1}}^\top + \textstyle\sum_j a_j b_j (0.5\mathbf{x}^{k+1} - \mathbf{z}_j){\mathbf{x}^{k+1}}^\top \leq \mathbf{x}^k\mathbf{Q}{\mathbf{x}^k}^\top + \mathbf{c}{\mathbf{x}^k}^\top + \textstyle\sum_j a_j b_j (0.5\mathbf{x}^k - \mathbf{z}_j){\mathbf{x}^k}^\top. \tag{A.2}$$

Second, for any pair of vectors $\{\mathbf{x}^k, \mathbf{x}^{k+1}\}$, w.r.t. any $\mathbf{z}_j$, we have,

$$0.5\|\mathbf{x}^k - \mathbf{z}_j\|_2^2 + 0.5\|\mathbf{x}^{k+1} - \mathbf{z}_j\|_2^2 \geq \|\mathbf{x}^k - \mathbf{z}_j\|_2 \|\mathbf{x}^{k+1} - \mathbf{z}_j\|_2. \tag{A.3}$$

We could further unfold Eq. (A.3) as follows.

$$\begin{aligned}
(0.5\mathbf{x}^k{\mathbf{x}^k}^\top - \mathbf{z}_j{\mathbf{x}^k}^\top + 0.5\mathbf{z}_j\mathbf{z}_j^\top) + 0.5\mathbf{z}_j\mathbf{z}_j^\top &\geq \|\mathbf{x}^k - \mathbf{z}_j\|_2\|\mathbf{x}^{k+1} - \mathbf{z}_j\|_2 - 0.5\mathbf{x}^{k+1}{\mathbf{x}^{k+1}}^\top + \mathbf{z}_j{\mathbf{x}^{k+1}}^\top, \\
\mathbf{x}^k{\mathbf{x}^k}^\top - 2\mathbf{z}_j{\mathbf{x}^k}^\top + \mathbf{z}_j\mathbf{z}_j^\top - 0.5\mathbf{x}^k{\mathbf{x}^k}^\top + \mathbf{z}_j{\mathbf{x}^k}^\top &\geq \|\mathbf{x}^k - \mathbf{z}_j\|_2\|\mathbf{x}^{k+1} - \mathbf{z}_j\|_2 - 0.5\mathbf{x}^{k+1}{\mathbf{x}^{k+1}}^\top + \mathbf{z}_j{\mathbf{x}^{k+1}}^\top, \\
\|\mathbf{x}^k - \mathbf{z}_j\|_2^2 - 0.5\mathbf{x}^k{\mathbf{x}^k}^\top + \mathbf{z}_j{\mathbf{x}^k}^\top &\geq \|\mathbf{x}^k - \mathbf{z}_j\|_2\|\mathbf{x}^{k+1} - \mathbf{z}_j\|_2 - 0.5\mathbf{x}^{k+1}{\mathbf{x}^{k+1}}^\top + \mathbf{z}_j{\mathbf{x}^{k+1}}^\top, \\
\|\mathbf{x}^k - \mathbf{z}_j\|_2\|\mathbf{x}^{k+1} - \mathbf{z}_j\|_2 - (0.5\mathbf{x}^{k+1} - \mathbf{z}_j){\mathbf{x}^{k+1}}^\top &\leq \|\mathbf{x}^k - \mathbf{z}_j\|_2^2 - (0.5\mathbf{x}^k - \mathbf{z}_j){\mathbf{x}^k}^\top.
\end{aligned}$$

Since $b_j = \frac{1}{\|\mathbf{x}^k - \mathbf{z}_j\|_2}$, we could multiply b_j on both sides of the above inequation, and have,

$$\|\mathbf{x}^{k+1} - \mathbf{z}_j\|_2 - b_j(0.5\mathbf{x}^{k+1} - \mathbf{z}_j)\mathbf{x}^{k+1\top} \leq \|\mathbf{x}^k - \mathbf{z}_j\|_2 - b_j(0.5\mathbf{x}^k - \mathbf{z}_j)\mathbf{x}^{k\top},$$

$$\textstyle\sum_j a_j\|\mathbf{x}^{k+1} - \mathbf{z}_j\|_2 - \sum_j a_j b_j(0.5\mathbf{x}^{k+1} - \mathbf{z}_j)\mathbf{x}^{k+1\top} \leq \sum_j a_j\|\mathbf{x}^k - \mathbf{z}_j\|_2 - \sum_j a_j b_j(0.5\mathbf{x}^k - \mathbf{z}_j)\mathbf{x}^{k\top}. \tag{A.4}$$

By summing up Eqs. (A.2) and (A.4), we get,

$$f(\mathbf{x}^{k+1}) \leq f(\mathbf{x}^k). \tag{A.5}$$

Based on this proved monotonicity, $f(\mathbf{x})$ is guaranteed to achieve the global optimum if and only if $\mathbf{x}^{k+1} = \mathbf{x}^k$ or $\mathbf{x} = \mathbf{z}_j$ since it is convex. □

We now prove Corollary 2.4.

Proof. Based on Theorem 2.3, it is straightforward to prove Corollary 2.4 by defining the corresponding known **Q**, **c**, and a_j, for all the $j \in N(i)$, as follows.

$$\mathbf{Q} = \mathbf{Z}^{t\top}\mathbf{Z}^t + 0.5\rho\mathbf{I}, \tag{A.6}$$

$$\mathbf{c} = \rho\mathbf{u}_i^t - \rho\mathbf{z}_i^t - 2\mathbf{s}_i\mathbf{Z}^t, \tag{A.7}$$

$$a_j = \lambda g_{ij}. \tag{A.8}$$

□

We now prove Theorem 3.1.

Proof. Assume that we walk from node $i \in \mathcal{V}$ to an arbitrary attribute category δ_k, then the process of traveling from δ_k to node j is independent of the one from i to δ_k. Therefore, we compute the probability of jumping from i to j as,

$$\begin{aligned} P(i \to j) &= \sum_{k=1}^{m} P(i \to \delta_k) P(\delta_k \to j), \\ &= \sum_{k=1}^{m} \bar{a}_{ik} \frac{\bar{a}_{jk}}{\sum_{q=1}^{n} \bar{a}_{qk}}, \\ &= [\bar{a}_{i1} d_{11}, \bar{a}_{i2} d_2, \ldots, \bar{a}_{im} d_{mm}] \bar{\mathbf{a}}_j^\top = s_{ij}. \end{aligned} \tag{A.9}$$

It should be noted that $P(i \to j) = P(j \to i)$ and $\sum_{i=1}^{n} s_{ij} = 1$. $\square$