Graph Recurrent Networks with Attributed Random Walks

Xiao Huang, Qingquan Song, Yuening Li, Xia Hu
Department of Computer Science and Engineering, Texas A&M University, College Station, TX, USA 77843
{xhuang,qqsong,liyuening,xiahu}@tamu.edu

ABSTRACT

Random walks are widely adopted in various network analysis tasks ranging from network embedding to label propagation. It could capture and convert geometric structures into structured sequences while alleviating the issues of sparsity and curse of dimensionality. Though random walks on plain networks have been intensively studied, in real-world systems, nodes are often not pure vertices, but own different characteristics, described by the rich set of data associated with them. These node attributes contain plentiful information that often complements the network, and bring opportunities to the random-walk-based analysis. 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.

To bridge the gap, we explore to perform joint random walks on attributed networks, and utilize them to boost the deep node representation learning. The proposed framework GraphRNA consists of two major components, i.e., a collaborative walking mechanism - AttriWalk, and a tailored deep embedding architecture for random walks, named graph recurrent networks (GRN). AttriWalk considers node attributes as a bipartite network and uses it to propel the walking more diverse and mitigate the tendency of converging to nodes with high centralities. AttriWalk enables us to advance the prominent deep network embedding model, graph convolutional networks, towards a more effective architecture - GRN. GRN empowers node representations to interact in the same way as nodes interact in the original attributed network. Experimental results on real-world datasets demonstrate the effectiveness of GraphRNA compared with the state-of-the-art embedding algorithms.

ACM Reference Format:
Xiao Huang, Qingquan Song, Yuening Li, Xia Hu. 2019. Graph Recurrent Networks with Attributed Random Walks. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3292500.3330941

1 INTRODUCTION

With the prevalence of networked systems, such as social media [35] and protein-protein interaction networks [10], network analysis has been widely applied in practice. Random walks, a fundamental tool, have been demonstrated to be effective in a variety of network analysis tasks, such as network embedding [9, 34], link prediction [1], and recommendations [32]. A random walk is a stochastic process that successively travels among nodes. It often serves as a key function that samples local topological structures and converts the complex node interactions into structured sequences. It has been proved that the vanilla random walks' transition probabilities are equivalent to the normalized cut and spectral segmentation under certain constraints [30, 40]. Since walking is naturally conducted in a distributed manner, random-walk-based methods enjoy the nice properties of being robust to sparsity and high dimensions [32]. However, limitations brought by random walks might also restrict the analysis performance. Vanilla random walks tend to converge to nodes with high centralities [28, 32], determined by their mathematical properties [30]. As verified in many applications empirically [5], this tendency would render the learning results to be less discriminative.

While random walks on plain networks have been intensively investigated [1, 34], in real-world systems, nodes are often not pure vertices, but associated with plentiful attribute data, aka node attributes, describing the particular characteristics of nodes. Such systems are called attributed networks [12, 22]. Examples include Twitter networks with tweets and academic networks with paper abstracts. Node attributes provide rich and highly related auxiliary information apart from network interactions for characterizing the node properties [36]. For instance, in Amazon co-purchasing networks, products purchased together tend to receive reviews with similar topics [29]. Massive reviews also serve as a reliable resource for customers to find the properties of products and make informed purchase decisions. These node attributes could potentially be used to advance the random-walk-based analysis. However, it is a nontrivial task to develop effective random walks on attributed networks towards a joint information extraction. Moreover, given the structured walking sequences learned from attributed networks, it is still challenging to take advantage of them for various off-the-shelf analysis algorithms such as SVM, while preserving the nice walking properties.

Among numerous network analysis techniques, deep network embedding architectures especially graph convolutional networks (GCN) have attracted wide attention [3, 17]. The learned node representations could preserve the original network information, and be directly employed as node features to advance various downstream applications such as node classification [17], network clustering [40], and anomaly detection [27]. 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 [10, 43] have attempted to exploit GCN to incorporate random walks into deep network embedding. However, they only adopt

KDD '19, August 4–8, 2019, Anchorage, AK, USA

ACM ISBN 978-1-4503-6201-6/19/08…$15.00
https://doi.org/10.1145/3292500.3330941

conventional random walks and the aggregation operation in GCN does not take the node order information into consideration [16].

We propose to perform effective random walks on attributed networks and convolve the extracted information via deep learning techniques for node representation learning. However, this remains a challenging task. First, node attributes are heterogeneous with respect to the topological structures [13]. Simply plugging them into existing random-walk-based models might burden the computation while downgrading the performance. Second, node attributes make the node interactions more complicated. To unravel the complex node interactions and nonlinear correlations in attributed networks, it is desired to have an appropriate joint random walk mechanism and an effective embedding algorithm. Third, 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.

To bridge this gap, we propose a novel attributed network embedding framework named **Graph R**ecurrent **N**etworks with **A**ttributed random walks (GraphRNA), consisting of an effective joint walking mechanism named AttriWalk, and tailored graph recurrent neural networks conjoining the advantage of recurrent neural architectures with AttriWalk. Specifically, we aim at investigating two research questions. (i) How to leverage the heterogeneous information, node attributes, to alleviate the limitations of vanilla random walks and conduct informative joint random walks? (ii) How to take advantage of the learned attributed random walks, and effectively model the complex node interactions and nonlinear correlations in attributed networks? We summarize our major contributions as follows.

- **GraphRNA**. We define a novel problem named random-walk-based attributed network embedding, and propose an effective framework - GraphRNA, to solve the problem.
- **Attributed Random Walks**. We design a joint walking mechanism AttriWalk, which considers node attributes as a bipartite network, and utilizes it to propel the random walks more diverse and mitigate the tendency of converging to nodes with high centralities.
- **Graph Recurrent Networks**. Based on AttriWalk, we advance graph convolutional networks to a more effective neural architecture, named graph recurrent networks, in which node representations are allowed to interact within the same way as nodes interact in the original attributed network.
- **Evaluation**. We empirically validate the effectiveness of GraphRNA on three real-world attributed networks.

2 PROBLEM STATEMENT

Notations: In this paper, we denote matrices with uppercase bold letters (e.g., $\mathbf{G}$), vectors with lowercase bold letters (e.g., $\mathbf{g}$), and scalars with lowercase alphabets (e.g., g). The transpose of a matrix or a vector is denoted as $\mathbf{G}^\top$ or $\mathbf{g}^\top$. We use $\mathbf{g}_i$ to represent the i^{th} row of $\mathbf{G}$, and g_{ij} to denote the element in the i^{th} row and j^{th} column of $\mathbf{G}$. 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 $\mathbf{z}$.

Table 1: The main symbols and definitions in this paper.

Notation	Definition
$n = \lvert\mathcal{V}\rvert \in \mathbb{R}$	total number of nodes
$m = \lvert\mathcal{U}\rvert \in \mathbb{R}$	number of node attribute categories
$d \in \mathbb{R}$	dimension of embedding representation
$\delta_j \in \mathcal{U}$	the j^{th} node attribute category
$\mathbf{G} \in \mathbb{R}_+^{n\times n}$	weighted adjacency matrix
$\mathbf{A} \in \mathbb{R}_+^{n\times m}$	node attribute matrix
$\mathbf{H} \in \mathbb{R}^{n\times d}$	final embedding representation
$\alpha \in \mathbb{R}_+$	probability of walking on network $\mathbf{G}$
$L \in \mathbb{R}_+$	length of each truncated random walk
$\mathcal{A}$	bipartite network constructed based on $\mathbf{A}$
$\mathcal{T}$	node index sequences sampled from $\mathbf{G}$ and $\mathcal{A}$
$\tau_i \in \mathcal{T}$	node sequences sampled for modeling node i

We summarize the main notations and their definitions in Table 1. Let $\mathcal{V}$ be a set of n nodes in a real-world information system, connected by an undirected network with the weighted adjacency matrix denoted as $\mathbf{G} \in \mathbb{R}^{n\times n}$. For each pair of nodes i and j, if there is no link between them, w_{ij} would be 0, and a larger w_{ij} indicates their relation tends to be stronger. Beyond the link relations, each node $i \in \mathcal{V}$ is also associated with a high-dimensional descriptive feature vector $\mathbf{a}_i$, known as node attributes. We use the matrix $\mathbf{A} \in \mathbb{R}^{n\times m}$ to collect all the node attributes. This type of network $\mathcal{G} = (\mathcal{V}, \mathbf{G}, \mathbf{A})$ is defined as an attributed network. To make the problem physically meaningful, we assume that elements of $\mathbf{G}$ and $\mathbf{A}$ are all non-negative.

Random walks have been widely adopted to model the topological structures in various network analysis tasks [1, 32]. While random-walk-based embedding over plain networks has been well studied [9, 34], node attributes bring opportunities to both random walks and embedding. Based on the aforementioned terminologies, we first formally define attributed network embedding (ANE) [8, 12], and then propose the random-walk-based ANE problem.

Definition 1 (Attributed Network Embedding) *Given an attributed network $\mathcal{G} = (\mathcal{V}, \mathbf{G}, \mathbf{A})$ and a predefined small dimension d, the goal is to learn a mapping $f : \{\mathbf{G}, \mathbf{A}\} \to \mathbf{H}$, where $\mathbf{H} \in \mathbb{R}^{n\times d}$, with each row $\mathbf{h}_i$ corresponding to the embedding representation of node $i \in \mathcal{V}$, such that the relation information described by $\mathbf{G}$ and the node attribute information characterized by $\mathbf{A}$ could be preserved as much as possible in $\mathbf{H}$. The effectiveness of this mapping f is evaluated based on the effectiveness of $\mathbf{H}$ when directly applied to real-world applications such as node classification.*

Attributed network embedding has attracted considerable attention recently since it serves as a bridge connecting real-world high-dimensional networked data and off-the-shelf analysis algorithms [11]. However, ANE remains a challenging task [13]. 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. We formally define this problem as follows.

Definition 2 (Random-walk-based Attributed Network Embedding) *The goal is to develop a framework that accords with the data characteristics of ANE, including complex node interactions,*

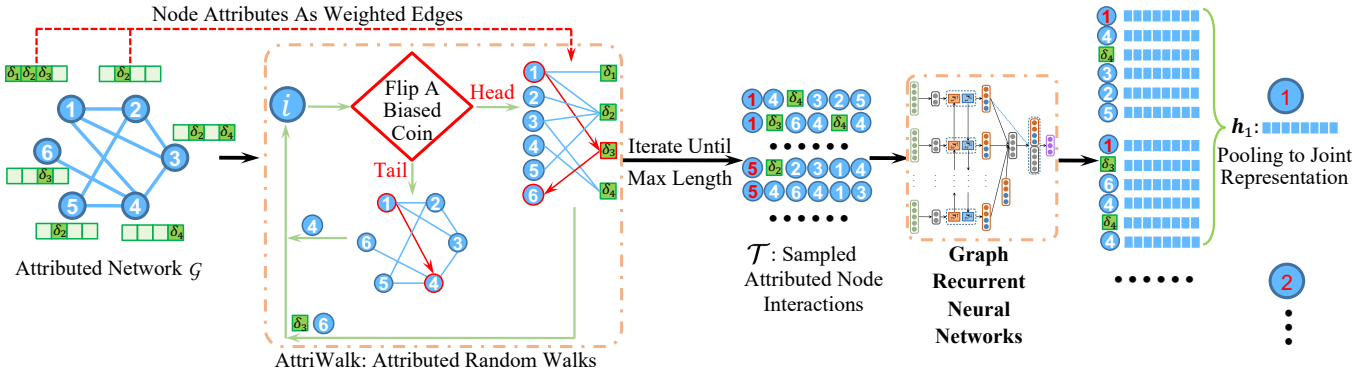


Figure 1: GraphRNA defines a unified walking mechanism AttriWalk, to enable joint random walks within network and node attributes, and a novel graph recurrent neural architecture to simulate attributed node interactions within the model.

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

3 ATTRIBUTED WALKS BASED EMBEDDING

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 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 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. We now introduce the details.

3.1 Attributed Random Walks - AttriWalk

In this section, 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 $\mathbf{A}$ [12, 25]. Jointly sampling $\mathcal{G}$ and $\mathbf{A}$ would make random walks more informative [36].

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 [20]. However, when directly applied to model topological structures, existing deep architectures often achieve suboptimal performance, given that networks are within irregular or non-Euclidean spaces [6]. While attempts have been made based on graph convolutional networks [16, 17] or sophisticated objective functions [8, 21, 25, 26, 44], 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.

3.1.1 Intuitive Solution. To jointly sample the network and node attribute information, an intuitive solution [12, 35] is to construct a new network $\mathbf{S}$ based on $\mathbf{A}$, in which the edge weight between nodes i and j is defined as the similarity between $\mathbf{a}_i$ and $\mathbf{a}_j$. In such a way, $\mathbf{A}$ is transformed into a topological structure $\mathbf{S}$. It is straightforward to conduct joint random walks on $\mathbf{G}$ and $\mathbf{S}$. However, this intuitive solution are expensive. The time complexity of computing $\mathbf{S}$ is $O(n\mathcal{N}_A)$, where $\mathcal{N}_A$ denotes the total number of nonzero elements in $\mathbf{A}$. In addition, in practice, as described by the power law, there would exist some high frequency attribute categories. They could make $\mathbf{S}$ dense. Thus, the calculation, storage, and walking on $\mathbf{S}$ would have high time and space complexity.

3.1.2 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 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,

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

(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,

$$P(i \to \delta_k) = \frac{\bar{a}_{ik}}{\sum_{p=1}^{m} \bar{a}_{ip}} = \bar{a}_{ik}. \tag{2}$$

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

$$P(\delta_k \to j) = \frac{\bar{a}_{jk}}{\sum_{q=1}^{n} \bar{a}_{qk}}. \tag{3}$$

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,

$$\mathbf{P} = \begin{bmatrix} \alpha\bar{\mathbf{G}} & (1-\alpha)\bar{\mathbf{A}} \\ (1-\alpha)\bar{\mathbf{A}}^\top & \mathbf{0} \end{bmatrix} \in \mathbb{R}_+^{(n+m)\times(n+m)}. \tag{4}$$

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

3.1.3 Theoretical Properties. AttriWalk defines unified walks within networks $\bar{\mathbf{G}}$ and $\mathcal{A}$. The walks on $\bar{\mathbf{G}}$ inherit the nice properties of traditional random walks [30, 34, 40], 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.

Theorem 3.1. *In AttriWalk, when the coin turns head, 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 $\mathbf{S}$ defined as follows.*

$$\mathbf{S} = \bar{\mathbf{A}}\mathbf{D}\bar{\mathbf{A}}^\top, \tag{5}$$

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

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,

$$\begin{aligned} P(i \to j) &= \sum_{k=1}^{m} P(i \to \delta_k) P(\delta_k \to j), \\ &= \sum_{k=1}^{m} \bar{a}_{ik} \frac{\bar{a}_{jk}}{\sum_{q=1}^{n} \bar{a}_{qk}}, \\ &= [\bar{a}_{i1}d_{11}, \bar{a}_{i2}d_{2}, \ldots, \bar{a}_{im}d_{mm}]\bar{\mathbf{a}}_j^\top \\ &= s_{ij}. \end{aligned} \tag{6}$$

□

We could use the alias method [7] to sample from a discrete probability distribution, and need to walk $O(n)$ steps in total. Comparing with the time complexity of the intuitive solution $O(n\mathcal{N}_A + n^2 + n)$, the time complexity of AttriWalk is $O(\mathcal{N}_A + n)$.

3.1.4 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 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 $\mathcal{T}$.

3.2 Graph Recurrent Networks - GRN

We now study to perform effective embedding by taking advantage of the learned sequences $\mathcal{T}$. A widely adopted approach [9, 34] is to consider the sequences as sentences and nodes as words, and apply word embedding techniques [31] 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 [10, 43] 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 [16, 17].

3.2.1 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 2, with details as follows.

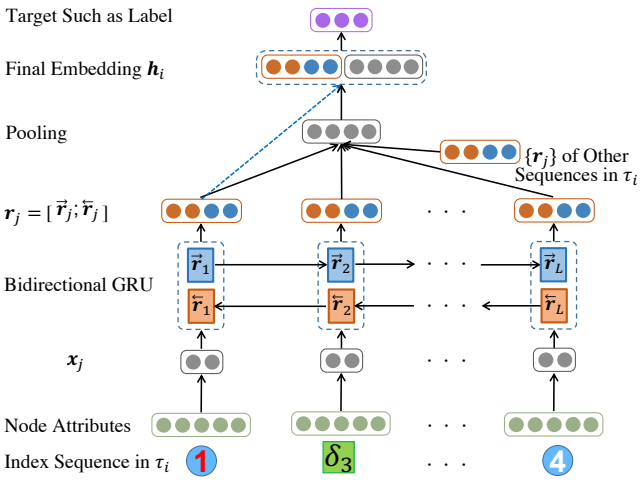


Figure 2: Architecture of the graph recurrent networks.

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,

$$\mathbf{x}_j = \sigma(\mathbf{a}_j \mathbf{W}_a + \mathbf{b}_a), \text{ or } \mathbf{x}_j = \sigma(\mathbf{e}_j \mathbf{W}_a + \mathbf{b}_a), \tag{7}$$

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 [2] 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. Mathematically, $\overrightarrow{\mathbf{r}}_j$ is calculated via,

$$\mathbf{f}_j = \sigma(\mathbf{x}_j \mathbf{W}_{xf} + \overrightarrow{\mathbf{r}}_{j-1} \mathbf{W}_{hf} + \mathbf{b}_f), \tag{8}$$

$$\mathbf{i}_j = \sigma(\mathbf{x}_j \mathbf{W}_{xi} + \overrightarrow{\mathbf{r}}_{j-1} \mathbf{W}_{hi} + \mathbf{b}_i), \tag{9}$$

$$\mathbf{o}_j = \sigma(\mathbf{x}_j \mathbf{W}_{xo} + \overrightarrow{\mathbf{r}}_{j-1} \mathbf{W}_{ho} + \mathbf{b}_o), \tag{10}$$

$$\mathbf{c}_j = \mathbf{f}_j \circ \mathbf{c}_{j-1} + \mathbf{i}_j \tanh(\mathbf{x}_j \mathbf{W}_{xc} + \overrightarrow{\mathbf{r}}_{j-1} \mathbf{W}_{hc} + \mathbf{b}_c), \tag{11}$$

$$\overrightarrow{\mathbf{r}}_j = \mathbf{o}_j \circ \tanh(\mathbf{c}_j). \tag{12}$$

$\mathbf{f}_j$, $\mathbf{i}_j$, 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 [10, 43] 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,

$$\mathbf{h}_i = [\hat{\mathbf{r}}_i, \bar{\mathbf{r}}_1]. \tag{13}$$

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.

3.2.2 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 [10, 16]. It could be considered as a first-order approximation of spectral graph convolutions [6, 17].

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 [4, 43] 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 [24]. Thus, nodes could only interact a few times within GCN. But in GRN, the bidirectional RNN layer allows much longer interactions.

Algorithm 1: GraphRNA

Input: $\mathbf{G}$, $\mathbf{A}$, d, B, L, α, and labels.
Output: Attributed network embedding representation $\mathbf{H}$.

```
Ḡ and Ā ← use ℓ1 norm to normalize each row of G and A;
Compute P based on Eq. (4);
Construct a probability table and an alias table for each p_i;
for Node i in V do
    for j = 1 : B do
        Set i as the initial node;
        Perform a (L − 1) length attributed random walks
          based on the probability tables and alias tables;
        Append the learn sequence to τ_i;
    Append τ_i to T;
for epoch = 1 : epoch_max do
    Shuffle the order of τ_i in T;
    for Set τ_i in T do
        Take all B sequences in τ_i as input and node i's label
          as output to train the GRN, as shown in Figure 2;
        Update weight matrices and bias terms to minimize
          the objective function L in Eq. (14);
for τ_i = τ_1 : τ_n do
    h_i ← Input all B sequences in τ_i into the trained GRN;
```

3.3 Optimization

GraphRNA could be trained with an unsupervised, supervised, or semisupervised setting. This nice property is inherit from GCN [10, 17]. 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,

$$\mathcal{L} = -\sum_{i \in \mathcal{V}} \mathbf{y}_i^\top \log(\text{softmax}(\sigma(\mathbf{h}_i \mathbf{W}_h + \mathbf{b}_h))). \tag{14}$$

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 function [10, 38].

We summarize the optimization of GraphRNA in Algorithm 1. 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 [7] 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. (14). After training the GRN over several epochs, we use it to embed τ_i into $\mathbf{h}_i$.

4 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.

- 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?
- GraphRNA mainly consists of attributed random walks and GRN. How much does each component contribute?
- 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 ?

4.1 Datasets

We employ three publicly available attributed networks in the experiments. Their statistical information is summarized in Table 2.

BlogCatalog [12]: It is a dataset of a blog community social network, which contains 5,196 users as nodes, 171,743 edges indicating the user interactions, and 8,189 attribute categories denoting the keywords of their blogs. Users could register their blogs into six different predefined classes, which are set as labels.

Flickr [12]: It is a benchmark attributed social network dataset containing 7,575 nodes. Each node is a Flickr user and each attribute category is a tag related to the photos shared by users. There are 239,738 undirected edges in this network, which indicate the following relationships among users. The nine groups that users have joined are considered as target labels.

Citation [39]: It is an attributed citation network consisting of 48,579 papers published in ACM. Words in the paper abstracts are adopted as node attributes based on the bag-of-words model. Citation links are treated as edges. Specifically, 10,000 distinct words and 119,974 citation edges exist in this datasets. Labels are defined as the areas that the topics of papers lie in.

4.2 Baseline Methods

To validate the effectiveness of GraphRNA, we include three categories of baselines as follows. 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.

Table 2: Statistics of the three real-world datasets.

	$\lvert\mathcal{V}\rvert$	# Edge	Density	$\lvert\mathcal{U}\rvert$	# Label
BlogCatalog	5,196	171,743	$1.27e{-}002$	8,189	6
Flickr	7,575	239,738	$8.36e{-}003$	12,047	9
Citation	48,579	119,974	$1.02e{-}004$	10,000	9

- *DeepWalk [34]:* It performs a stream of truncated vanilla random walks on the network, and employs word2vec [31] to embed the sampled sequences to learn $\mathbf{H}$.
- *LINE [38]:* It learns $\mathbf{H}$ from the network structure by modeling both the one-hop and two-hop neighbors of each node.
- *Attribute-Spec [40]:* 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.
- *MultiSpec [18]:* It first computes node similarities based on $\mathbf{G}$ and $\mathbf{A}$ independently to construct two graphs, and jointly embeds them via a coupled spectral embedding.
- *AANE [11]:* It updates $\mathbf{H}$ in a distributed way, based on the decomposition of local node attribute affinity and the embedding difference between connected nodes.
- *GCN [17]:* The vanilla GCN. It learns $\mathbf{H}$ based on the first-order approximation of spectral graph convolutions.
- *GraphSAGE [10]:* It advances GCN via introducing a set of aggregator functions that learn to aggregate node attributes from a node's local neighbors.

4.3 Experimental Settings

We follow the conventional settings [3, 41] to validate the performance of GraphRNA. The effectiveness of an embedding method is evaluated by applying its learned latent representations $\mathbf{H}$ to perform a node classification task. In this task, given the embedding representations of all nodes $\mathcal{V}$, and a set of training nodes with labels, the goal is to predict the labels of the remaining nodes in $\mathcal{V}$. The classification performance is measured by two standard metrics, i.e., micro-average and macro-average [12].

Given the learned $\mathbf{H}$, five-fold cross-validation is employed to select the training and test sets. For each dataset, given the entire nodes $\mathcal{V}$, we randomly select 80% of $\mathcal{V}$ as the training set, and use the remaining as the test set. Within the training set, we randomly select 10% from it and set as the validation set. In such a way, we have separated the learned $\mathbf{H}$ into $\mathbf{H}_{\text{train}}$, $\mathbf{H}_{\text{test}}$, and $\mathbf{H}_{\text{valid}}$. To conduct node classification, we leverage $\mathbf{H}_{\text{train}}$ and the corresponding labels to train an SVM classifier, and apply it to predict the labels of nodes in the test set, based on $\mathbf{H}_{\text{test}}$. 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 hyper-parameters, 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.

Table 3: The node classification performance of all methods on all the three attributed networks in terms of micro-average.

Training Set Used For Evaluation		BlogCatalog			Flickr			Citation		
		25%	50%	100%	25%	50%	100%	25%	50%	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(+1.8%)	0.332	0.421	0.516(−3.6%)	0.549	0.624	0.693(+1.3%)
	Attribute-Spec	0.791	0.841	0.869(+29.3%)	0.771	0.813	0.846(+58.1%)	0.688	0.700	0.704 (+2.9%)
Shallow ANE	MultiSpec	0.788	0.849	0.896(+33.3%)	0.720	0.800	0.859(+60.6%)	0.709	0.719	N.A.
	AANE	0.878	0.913	0.932(+38.7%)	**0.811**	0.854	0.885(+65.4%)	0.701	0.715	0.722(+5.6%)
Deep ANE	GCN-hidden	0.657	0.701	0.734(+9.2%)	0.521	0.558	0.588(+9.9%)	0.754	0.766	0.789(+15.4%)
	GraphSAGE-hidden	0.801	0.876	0.917(+36.5%)	0.612	0.743	0.841(+57.2%)	0.727	0.769	0.785(+14.8%)
	GraphRNA-hidden	0.857	0.903	0.938(+39.6%)	0.705	0.817	0.881(+64.7%)	0.727	0.755	0.781(+14.2%)
End-to-End	GCN	0.683	0.716	0.740(+10.1%)	0.514	0.563	0.590(+10.3%)	**0.763**	**0.776**	**0.798**(+16.7%)
	GraphSAGE	0.820	0.890	0.924(+37.5%)	0.626	0.757	0.852(+59.3%)	0.741	0.769	0.785(+14.8%)
	GraphRNA-noAttriW	0.865	0.909	0.942(+40.2%)	0.645	0.777	0.874(+63.4%)	0.746	0.768	0.788(+15.2%)
	GraphRNA-noRNN	0.874	0.920	0.943(+40.3%)	0.769	0.855	0.902(+68.6%)	0.739	0.768	0.784(+14.6%)
	GraphRNA	**0.885**	**0.925**	**0.948**(+41.1%)	0.776	**0.866**	**0.905**(+69.2%)	0.747	0.768	0.790(+15.5%)

4.4 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 3. Similar results are observed based on macro-average scores, so we omit them.

From the results in Table 3, we have three major observations. First, GraphRNA achieves the best performance on datasets BlogCatalog and Flickr, in terms of micro-average. On Citation, 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.

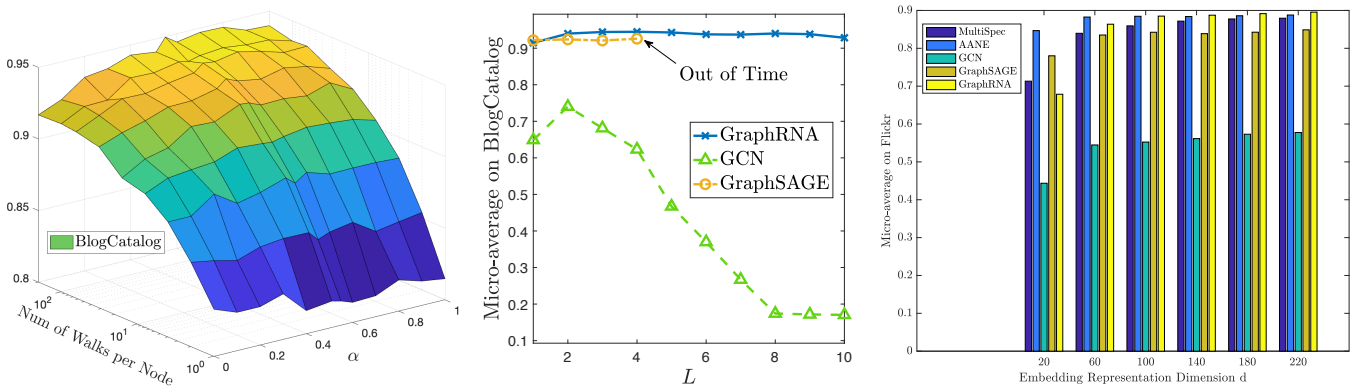
For each dataset, we vary the percentage of the training set that has been used for learning the classifiers as {25%, 50%, 100%}. It means that only {25%, 50%, 100%} of the four folds selected by the five-fold cross-validation, have been used. From the results in Table 3, we have similar observations 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.

4.5 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 3.

- *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 3, 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 [15].



(a) α and number of walks per node B **(b) Interaction depth L** **(c) Embedding representation dimension d**

Figure 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.

4.6 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 3a. Similar results are observed on other datasets, so we omit them. Form the results in Figures 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 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 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 embedding representation dimension d is interpreted as the number of hidden units in the layer before last target layer, as shown in Figure 2. We vary it from 20 to 220, and the performance of all ANE methods is shown in Figures 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 RELATED WORK

Attributed network embedding bridges the gap between real-world networked systems and downstream analysis algorithms. We roughly categorize existing ANE models into two classes as follows.

Shallow Attributed Network Embedding. To incorporate the topological structure and node attributes, attempts have been made from different aspects [12, 14]. Qi et al. [35] focused on multimedia objects and learned their latent semantic representations via jointly modeling their content information similarities and contextual links. Le and Lauw [19] advanced the topic modeling by incorporating the links between documents. Yang et al. [41] converted the network to pointwise mutual information and employed matrix tri-factorization to jointly embed the attributed network. Pan et al. [33] designed a coupled random walk based model with three objectives to incorporate the network, node attributes, and labels. Huang et al. [11] accelerated the ANE via decomposing it into many independent sub-problems and optimizing in a distributed way. Li et al. [22] designed an online ANE algorithm to tackle dynamic scenarios. Yang et al. [42] proposed to leverage the Weisfeiler-Lehman graph kernels to learn binary vector representations of attributed networks. Several coupled matrix factorization based ANE methods [23, 45] 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. [3] involved nonlinear multilayered embedding functions to collaboratively embed the heterogeneous network and node content. Kipf and Welling [16, 17] 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. [10] proposed the inductive representation learning, in which each node's representation is learned via aggregating all its neighbors' representations. Liang et al. [25] 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 [8, 21, 26, 44]. Recently, efforts have also been devoted

to leveraging graph recurrent neural networks to conduct node or graph representation embedding [15, 37].

6 CONCLUSION AND FUTURE WORK

Random walks on plain graphs have been intensively studied in network analysis. Despite the prominence it has shown, rare explorations are conducted on developing random-walk-based techniques for attributed networks to encode heterogenous information thus enhance node representation learning. To bridge this gap, 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.

For future work, as the graph recurrent networks are naturally applicable for sequential data, we intend to develop an online update scheme for encoding temporal drifting information in dynamic networks. Another valuable direction would be exploring other meaningful pooling methods besides the mean pooling adopted in this paper for joint representation.

7 ACKNOWLEDGMENTS

This work is, in part, supported by DARPA (#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.

REFERENCES

[1] Lars Backstrom and Jure Leskovec. 2011. Supervised Random Walks: Predicting and Recommending Links in Social Networks. In *WSDM*. 635–644.
[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *ICLR*.
[3] Shiyu Chang, Wei Han, Jiliang Tang, Guo-Jun Qi, Charu C. Aggarwal, and Thomas S. Huang. 2015. Heterogeneous Network Embedding via Deep Architectures. In *KDD*. 119–128.
[4] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *ICLR*.
[5] Colin Cooper, Sang Hyuk Lee, Tomasz Radzik, and Yiannis Siantos. 2014. Random Walks in Recommender Systems: Exact Computation and Simulations. In *WWW*. 811–816.
[6] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *NIPS*. 3844–3852.
[7] Luc Devroye. 1986. Sample-Based Non-Uniform Random Variate Generation. In *Winter Simulation Conference*. 260–265.
[8] Hongchang Gao and Heng Huang. 2018. Deep Attributed Network Embedding. In *IJCAI*. 3364–3370.
[9] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *KDD*. 855–864.
[10] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*. 1024–1034.
[11] Xiao Huang, Jundong Li, and Xia Hu. 2017. Accelerated Attributed Network Embedding. In *SDM*. 633–641.
[12] Xiao Huang, Jundong Li, and Xia Hu. 2017. Label Informed Attributed Network Embedding. In *WSDM*. 731–739.
[13] Xiao Huang, Jundong Li, Na Zou, and Xia Hu. 2018. A General Embedding Framework for Heterogeneous Information Learning in Large-Scale Networks. *TKDD* 12 (2018).
[14] Xiao Huang, Qingquan Song, Jundong Li, and Xia Hu. 2018. Exploring Expert Cognition for Attributed Network Embedding. In *WSDM*. 270–278.
[15] Yu Jin and Joseph F. JaJa. 2018. Learning Graph-Level Representations with Gated Recurrent Neural Networks. *arXiv preprint arXiv:1805.07683* (2018).
[16] Thomas N. Kipf and Max Welling. 2016. Variational Graph Auto-Encoders. In *NIPS Workshop on Bayesian Deep Learning*.
[17] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.
[18] Abhishek Kumar, Piyush Rai, and Hal Daume. 2011. Co-regularized Multi-view Spectral Clustering. In *NIPS*. 1413–1421.
[19] Tuan M. V. Le and Hady W. Lauw. 2014. Probabilistic Latent Document Network Embedding. In *ICDM*. 270–279.
[20] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep Learning. *Nature* 521, 436–444 (2015).
[21] Jihwan Lee and Sunil Prabhakar. 2018. A3embed: Attribute Association Aware Network Embedding. In *WWW*. 1243–1251.
[22] Jundong Li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. 2017. Attributed Network Embedding for Learning in a Dynamic Environment. In *CIKM*. 387–396.
[23] Juan-Hui Li, Chang-Dong Wang, Ling Huang, Dong Huang, Jian-Huang Lai, and Pei Chen. 2018. Attributed Network Embedding with Micro-Meso Structure. In *DASFAA*. 20–36.
[24] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning. In *AAAI*. 3538–3545.
[25] Jiongqian Liang, Peter Jacobs, Jiankai Sun, and Srinivasan Parthasarathy. 2018. Semi-Supervised Embedding In Attributed Networks With Outliers. In *SDM*. 153–161.
[26] Lizi Liao, Xiangnan He, Hanwang Zhang, and Tat-Seng Chua. 2018. Attributed Social Network Embedding. *TKDE* 30, 12 (2018), 2257–2270.
[27] Ninghao Liu, Xiao Huang, and Xia Hu. 2017. Accelerated Local Anomaly Detection via Resolving Attributed Networks. In *IJCAI*. 2337–2343.
[28] Naoki Masuda, Mason A Porter, and Renaud Lambiotte. 2017. Random Walks And Diffusion On Networks. *Physics Reports* 716 (2017), 1–58.
[29] Julian McAuley, Rahul Pandey, and Jure Leskovec. 2015. Inferring Networks of Substitutable and Complementary Products. In *KDD*. 785–794.
[30] Marina Meila and Jianbo Shi. 2001. A Random Walks View of Spectral Segmentation. (2001).
[31] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and Their Compositionality. In *NIPS*. 3111–3119.
[32] Athanasios N. Nikolakopoulos and George Karypis. 2019. RecWalk: Nearly Uncoupled Random Walks for Top-N Recommendation. In *WSDM*.
[33] Shirui Pan, Jia Wu, Xingquan Zhu, Chengqi Zhang, and Yang Wang. 2016. Tri-Party Deep Network Representation. In *IJCAI*. 1895–1901.
[34] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online Learning of Social Representations. In *KDD*. 701–710.
[35] Guo-Jun Qi, Charu Aggarwal, Qi Tian, Heng Ji, and Thomas S. Huang. 2012. Exploring Context and Content Links in Social Media: A Latent Space Method. *TPAMI* 34, 5 (2012), 850–862.
[36] Andrew N. Smith, Eileen Fischer, and Chen Yongjian. 2012. How does Brand-Related User-Generated Content Differ Across YouTube, Facebook, and Twitter? *Journal of Interactive Marketing* 26, 2 (2012), 102–113.
[37] Aynaz Taheri, Kevin Gimpel, and Tanya Berger-Wolf. 2018. Learning Graph Representations with Recurrent Neural Network Autoencoders. In *Proc. KDD Deep Learn. Day*.
[38] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale Information Network Embedding. In *WWW*. 1067–1077.
[39] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. Arnetminer: Extraction and Mining of Academic Social Networks. In *KDD*. 990–998.
[40] Ulrike von Luxburg. 2007. A Tutorial on Spectral Clustering. *Statistics and Computing* 17, 4 (2007), 395–416.
[41] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y. Chang. 2015. Network Representation Learning with Rich Text Information. In *IJCAI*. 2111–2117.
[42] Hong Yang, Shirui Pan, Peng Zhang, Ling Chen, Defu Lian, and Chengqi Zhang. 2018. Binarized Attributed Network Embedding. In *ICDM*.
[43] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *KDD*. 974–983.
[44] Zhen Zhang, Hongxia Yang, Jiajun Bu, Sheng Zhou, Pinggang Yu, Jianwei Zhang, Martin Ester, and Can Wang. 2018. ANRL: Attributed Network Representation Learning via Deep Neural Networks. In *IJCAI*. 3155–3161.
[45] Shenghuo Zhu, Kai Yu, Yun Chi, and Yihong Gong. 2007. Combining Content and Link for Classification Using Matrix Factorization. In *SIGIR*. 487–494.