BiNE: Bipartite Network Embedding

Ming Gao
School of Data Science and Engineering
East China Normal University
Shanghai, China
mgao@dase.ecnu.edu.cn

Leihui Chen
School of Data Science and Engineering
East China Normal University
Shanghai, China
leihuichen@gmail.com

Xiangnan He*
School of Computing
National University of Singapore
Singapore
xiangnanhe@gmail.com

Aoying Zhou
School of Data Science and Engineering
East China Normal University
Shanghai, China
ayzhou@dase.ecnu.edu.cn

ABSTRACT

This work develops a representation learning method for bipartite networks. While existing works have developed various embedding methods for network data, they have primarily focused on homogeneous networks in general and overlooked the special properties of bipartite networks. As such, these methods can be suboptimal for embedding bipartite networks.

In this paper, we propose a new method named BiNE, short for *Bipartite Network Embedding*, to learn the vertex representations for bipartite networks. By performing biased random walks purposefully, we generate vertex sequences that can well preserve the long-tail distribution of vertices in the original bipartite network. We then propose a novel optimization framework by accounting for both the explicit relations (i.e., observed links) and implicit relations (i.e., unobserved but transitive links) in learning the vertex representations. We conduct extensive experiments on several real datasets covering the tasks of link prediction (classification), recommendation (personalized ranking), and visualization. Both quantitative results and qualitative analysis verify the effectiveness and rationality of our BiNE method.

CCS CONCEPTS

• **Information systems** → **Information retrieval**; **Recommender systems**; • **Computing methodologies** → **Neural networks**;

KEYWORDS

Bipartite networks, Network representation learning, Link prediction, Recommendation

*Corresponding author.

SIGIR '18, July 8–12, 2018, Ann Arbor, MI, USA

ACM ISBN 978-1-4503-5657-2/18/07…$15.00
https://doi.org/10.1145/3209978.3209987

ACM Reference Format:
Ming Gao, Leihui Chen, Xiangnan He, and Aoying Zhou. 2018. BiNE: Bipartite Network Embedding. In *SIGIR '18: The 41st International ACM SIGIR Conference on Research and Development in Information Retrieval, July 8–12, 2018, Ann Arb1 or, MI, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3209978.3209987

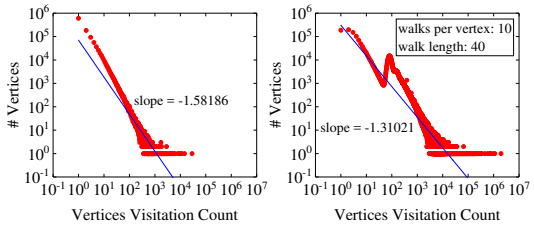
1 INTRODUCTION

The bipartite network is a ubiquitous data structure to model the relationship between two types of entities. It has been widely used in many applications such as recommender systems, search engines, question answering systems and so on. For example, in search engines, queries and webpages form a bipartite network, where the edges can indicate users' click behaviors that provide valuable relevance signal [1, 2]; in another application of recommender systems, users and items form a bipartite network, where the edges can encode users' rating behaviors that contain rich collaborative filtering patterns [3].

To perform predictive analytics on network data, it is crucial to first obtain the representations (i.e., feature vectors) for vertices. Traditional vector space methods such as the bag-of-words representations capture too few semantics and are inefficient to deal with large-scale dynamic networks in practical applications. Recent advances in data mining and information retrieval have focused on learning representations from data [4–7]. In particular, they embed vertices into a low dimensional space, i.e., representing a vertex as a learnable embedding vector. Based on the vertex embeddings, standard machine learning techniques can be applied to address various predictive tasks such as vertex labeling, link prediction, clustering and so on.

To date, existing works have primarily focused on embedding homogeneous networks where vertices are of the same type [4, 8–10]. Following the pioneering work of DeepWalk [8], these methods typically apply a two-step solution: first performing random walks on the network to obtain a "corpus" of vertices, and then applying word embedding methods such as word2vec [11] to obtain the embeddings for vertices. Despite effectiveness and prevalence, we argue that these methods can be suboptimal for embedding bipartite networks due to two primary reasons:

(1) The type information of vertices is not considered. Distinct from homogeneous networks, there are two types of vertices



(a) YouTube (b) Random walk generator

Figure 1: The vertex distribution of (a) the real-world YouTube dataset and (b) the corpus generated by the random walk generator of DeepWalk. The generated corpus does not show the desired power-law distribution due to the improper design of the generator.

in a bipartite network. Although edges exist between vertices of different types only, there are essentially implicit relations between vertices of the same type. For example, in the user-item bipartite network built for recommendation, there exists an implicit relation between users which can indicate their preference in consuming the same item; and importantly, it is recently reported that modeling such implicit relations can improve the recommendation performance [12]. However, existing network embedding methods modeled the explicit relation (i.e., observed edges) only and ignored the underlying implicit relations. While the corpus generated by random walks may capture such high-order implicit relations to a certain extent, we argue that a more effective way is to encode such implicit relations into representation learning in an explicit manner.

(2) The generated corpus may not preserve the characteristics of a bipartite network. To demonstrate this point, we plot the frequency distribution of vertices in a real YouTube dataset[1] in Figure 1(a). We can see that the vertices exhibit a standard power-law distribution with a slope of -1.582. By contrast, we plot the frequency distribution of vertices in a corpus generated by DeepWalk in Figure 1(b). We find that the generated distribution differs significantly from the real distribution, and it cannot be well described by a power-law distribution. We point out that the failure of DeepWalk is due to the improper design of the random walk generator, which is suboptimal for embedding bipartite networks. Specifically, it generates the same number of random walks starting from each vertex and constrains the length of walks to be the same; this limits the capability of the generator and makes it difficult to generate a corpus following a power-law distribution — which is a common characteristics of many real-world bipartite networks [13].

To our knowledge, none of the existing works has paid special attention to embed bipartite networks. While a recent work by Dong et al. [14] proposed metapath2vec++ for embedding heterogeneous networks which can also be applied to bipartite networks, we argue that a key limitation is that it treats the explicit and implicit relations as contributing equally to the learning. In real-world bipartite networks, the explicit and implicit relations typically carry different semantics. As such, they should be treated differently and assigned to varying weights in learning the vertex embeddings. This can be evidenced by existing recommendation works [15] that usually assign varying weights on different sources of information to allow a flexible tuning on the learning process.

In this work, we focus on the problem of learning vertex representations for bipartite networks. We propose BiNE (short for ***Bi**partite **N**etwork **E**mbedding*), which addresses the aforementioned limitations of existing network embedding methods. Below we highlight two characteristics of our BiNE method.

(1) To account for both the explicit relations and implicit relations, we propose a joint optimization framework. For each relation, we design a dedicated objective function; by sharing the vertex embeddings, the objective functions for different relations reinforce each other and lead to better vertex embeddings. Specifically, the modeling of the explicit relations aims to reconstruct the bipartite network by focusing on observed links. For the modeling of implicit relations, we aim to capture the high-order correlations in the bipartite network. To avoid the explosive growth of complexity in expanding a network, we similarly resort to performing random walks and design the objective function based on the generated corpora.

(2) To retain the properties of the bipartite network as many as possible, we propose a biased and self-adaptive random walk generator. Specifically, we set the number of random walks starting from each vertex based on its importance, making the vertex distribution in the generated corpus more consistent with the original bipartite network. Moreover, instead of setting a uniform length for all random walks, we allow a walk to be stopped in a probabilistic way. Through this way, we can generate vertex sequences of varying lengths, which is more analogous to the sentences in natural language. Our empirical study shows that our generator can generate corpus more close to the distribution of the real-world networks.

The remainder of the paper is organized as follows. We first review related work in Section 2. We formulate the problem in Section 3, before delving into details of the proposed method in Section 4. We perform extensive empirical studies in Section 5 and conclude the paper in Section 6.

2 RELATED WORK

2.1 Network Representation Learning

Our work is related to vertex representation learning methods on homogeneous networks, which can be categorized into two types: matrix factorization (MF)-based and neural network-based methods.

MF-based methods are either linear [16] or nonlinear [17] in learning vertex embeddings. The former employs the linear transformations to embed network vertices into a low dimensional embedding space, such as singular value decomposition (SVD) and multiple dimensional scaling (MDS) [16]. However, the latter maps network vertices into a low dimensional latent space by utilizing the nonlinear transformations, e.g., kernel PCA, spectral embedding , marginal fisher analysis (MFA), and manifold learning approaches include LLE and ISOMAP [17]. Generally speaking, MF-based methods have two main drawbacks: (1) they are usually computationally expensive due to the eigen-decomposition operations on data matrices, making them difficult to handle large-scale networks [18, 19];

[1]This YouTube dataset contains 1 million videos and 5 million links, which are downloaded from: http://socialnetworks.mpi-sws.org/data-imc2007.html

(2) their performance are rather sensitive to the predefined proximity measures for calculating the affinity matrix.

Neural network-based methods are the state-of-art vertex representation learning techniques. The pioneer work DeepWalk [8] and Node2vec [4] extend the idea of Skip-gram [11] to model homogeneous network, which is convert to a corpus of vertex sequences by performing truncated random walks. However, they may not be effective to preserve both explicit and implicit relations of the network. There are some follow-up works exploiting both 1st-order and 2nd-order proximities between vertices to embed homogeneous networks. Specifically, LINE [20] learns two separated embeddings for 1st-order and 2nd-order relations; SDNE [21] incorporates both 1st-order and 2nd-order proximities to preserve the network structure; and GraRep [22] further extends the method to capture higher-order proximities. Besides capturing high-order proximities, there are several proposals to incorporate side information into vertex embedding learning, such as vertex labels [10, 23], community information [24], textual content [25], user profiles [9], location information [26], among others.

It is worth pointing out that the above mentioned methods are designed for embedding homogeneous networks, for which there is only one type of vertices. In addition, the "corpus" generated by the truncated random walks may not capture the characteristics of the network structure, such as the power-law distribution of vertex degrees. Thus, these homogeneous network embedding methods might be suboptimal for learning vertex representations for a bipartite network.

Metapath2vec++ [14], HNE [27] and EOE [28] are representative vertex embedding methods for heterogeneous networks. Although they can be applied to bipartite network which can be seen as a special type of heterogeneous networks, they are not tailored for learning on bipartite networks. Specifically, HNE aims to integrate content and linkage structures into the embedding process, and Metapath2vec++ ignores the strength of the relations between vertices and treats the explicit and implicit relations as equally. As such, they are suboptimal for vertex representation learning for a bipartite network.

2.2 Bipartite Network Modeling

As a ubiquitous data structure, bipartite networks have been mined for many applications, among which vertex ranking is an active research problem. For example, HITS [29] learns to rank vertices by capturing some semantic relations within a bipartite network. Co-HITS [1] incorporates content information of vertices and the constraints on relevance into vertex ranking of bipartite network. BiRank [3] ranks vertices by taking into account both the network structure and prior knowledge.

Distributed vertex representation is an alternative way to leverage signals from bipartite network. Unlike the ranking task, it learns a low dimensional representation of a vertex, which can be seen as the "features" of the vertex that preserves more information rather than simply a ranking score. Latent factor model (LFM), which has been widely investigated in the field of recommender systems and semantic analysis, is the most representative model. And a typical implementation of LFM is based on matrix factorization [30–32]. Recent advances utilize deep learning methods to learn vertex embeddings on the user-item network for recommendation [33].

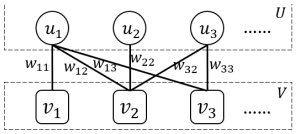


Figure 2: An example of the bipartite network structure

It is worth pointing out that these methods are tailored for the recommendation task, rather than for learning informative vertex embeddings. Moreover, they model the explicit relations in bipartite network only, which can be improved by incorporating implicit relations as shown in [12, 15].

3 PROBLEM FORMULATION

We first give notations used in this paper, and then formalize the bipartite network embedding problem to be addressed.

Notations. Let $G = (U, V, E)$ be a bipartite network, where U and V denote the set of the two types of vertices respectively, and $E \subseteq U \times V$ defines the inter-set edges. As shown in Figure 2, u_i and v_j denote the i-th and j-th vertex in U and V, respectively, where $i = 1, 2, ..., |U|$ and $j = 1, 2, ..., |V|$. Each edge carries a non-negative weight w_{ij}, describing the strength between the connected vertices u_i and v_j; if u_i and v_j are disconnected, the edge weight w_{ij} is set to zero. Therefore, we can use a $|U| \times |V|$ matrix $\mathbf{W} = [w_{ij}]$ to represent all weights in the bipartite network.

Problem Definition. The task of bipartite network embedding aims to map all vertices in the network into a low-dimensional embedding space, where each vertex is represented as a dense embedding vector. In the embedding space, both the implicit relations between vertices of the same type and the explicit relations between vertices of different types should be preserved. Formally, the problem can be defined as:

Input: A bipartite network $G = (U, V, E)$ and its weight matrix $\mathbf{W}$.
Output: A map function $f : U \cup V \rightarrow \mathbb{R}^d$, which maps each vertex in G to a d-dimensional embedding vector.

To keep the notations simple, we use $\vec{\mathbf{u}}_i$ and $\vec{\mathbf{v}}_j$ to denote the embedding vectors for vertices u_i and v_j, respectively. As such, we can present the embedding vectors of all vertices in the bipartite network as two matrices $\mathbf{U} = [\vec{\mathbf{u}}_i]$ and $\mathbf{V} = [\vec{\mathbf{v}}_j]$.

4 BINE: BIPARTITE NETWORK EMBEDDING

A good network embedding should be capable of reconstructing the original network well. To achieve this aim for a bipartite network, we consider reconstructing the bipartite network from two perspectives — the explicit relations evidenced by the observed edges and the implicit relations implied by the unobserved but transitive links. We then learn vertex embeddings by jointly optimizing the two tasks. This section presents our BiNE method along this line.

4.1 Modeling Explicit Relations

In a bipartite network, edges exist between vertices of two different types, providing an explicit signal on constructing the bipartite network. Similar to the modeling of 1st-order proximity in LINE [20], we model explicit relations by considering the local proximity between two connected vertices. The joint probability between vertices u_i and v_j is defined as:

$$P(i, j) = \frac{w_{ij}}{\sum_{e_{ij} \in E} w_{ij}}. \quad (1)$$

where w_{ij} is the weight of edge e_{ij}. Obviously, if two vertices are strongly connected with a larger weight, they will have a higher probability to be co-occurred.

Now we consider how to estimate the local proximity between two vertices in the embedding space. The effectiveness and prevalence of word2vec inspire many works [4, 8, 20] to use inner product to model the interaction between two entities. We follow this setting, and use sigmoid function to transform the interaction value to the probability space:

$$\hat{P}(i,j) = \frac{1}{1 + exp(-\vec{\mathbf{u}}_i^T \vec{\mathbf{v}}_j)}. \tag{2}$$

where $\vec{\mathbf{u}}_i \in \mathbb{R}^d$ and $\vec{\mathbf{v}}_j \in \mathbb{R}^d$ are the embedding vectors of vertices u_i and v_j, respectively.

With the empirical distribution of the co-occurring probability between vertices and the reconstructed distribution, we can learn the embedding vectors by minimizing their difference. We choose the KL-divergence as the difference measure between distributions, which can be defined as:

$$\begin{aligned} minimize \quad O_1 = KL(P||\hat{P}) &= \sum_{e_{ij} \in E} P(i,j) \log(\frac{P(i,j)}{\hat{P}(i,j)}) \\ &\propto -\sum_{e_{ij} \in E} w_{ij} \log \hat{P}(i,j). \end{aligned} \tag{3}$$

Intuitively, minimizing this objective function will make two vertices that are strongly connected in the original network also close with each other in the embedding space, which preserves the local proximity as desired.

4.2 Modeling Implicit Relations

As illustrated in existing recommedation works [12, 15], both explicit and implicit relations are helpful to reveal different semantic in bipartite networks. To be comprehensive, it is crucial to also account for the implicit relation between two vertices of the same type, even though they are not explicitly connected. Intuitively, for two vertices of the same type, if there exists a path between them, there should be certain implicit relation between them; the number of the paths and their length indicate the strength of the implicit relation. Unfortunately, counting the paths between two vertices has a rather high complexity of an exponential order, which is infeasible to implement for large networks. To encode such high-order implicit relations among vertices in a bipartite network, we resort to the solution of DeepWalk. Specifically, the bipartite network is first converted to two corpora of vertex sequences by performing random walks; then the embeddings are learned from the corpora which encodes high-order relations between vertices. In what follows, we first elaborate how to generate two quality corpora for a bipartite network.

4.2.1 ***Constructing Corpus of Vertex Sequences.*** It is a common way to convert a network into a corpus of vertex sequences by performing random walks on the network, which has been used in some homogeneous network embedding methods [4, 8]. However, directly performing random walks on a bipartite network could fail, since there is no stationary distribution of random walks on bipartite networks due to the periodicity issue [34]. To address this issue, we consider performing random walks on two homogeneous networks that contain the 2nd-order proximity between vertices of the same type. Following the idea of Co-HITS [1], we define the 2nd-order proximity between two vertices as:

$$w_{ij}^U = \sum_{k \in V} w_{ik} w_{jk}; \quad w_{ij}^V = \sum_{k \in U} w_{ki} w_{kj}. \tag{4}$$

where w_{ij} is the weight of edge e_{ij}. Hence, we can use the $|U| \times |U|$ matrix $\mathbf{W}^U = [w_{ij}^U]$ and the $|V| \times |V|$ matrix $\mathbf{W}^V = [w_{ij}^V]$ to represent the two induced homogeneous networks, respectively.

Now we can perform truncated random walks on the two homogeneous networks to generate two corpora for learning the high-order implicit relations. As demonstrated in Figure 1, the corpus generated by DeepWalk may not capture the characteristic of a real-world network. To generate a corpus with a high fidelity, we propose a biased and self-adaptive random walk generator, which can preserve the vertex distribution in a bipartite network. We highlight its core designs as follows:

- First, we relate the number of random walks starting from each vertex to be dependent on its importance, which can be measured by its centrality. For a vertex, the greater its centrality is, the more likely a random walk will start from it. As a result, the vertex importance can be preserved to some extent.
- We assign a probability to stop a random walk in each step. In contrast to DeepWalk and other work [14] that apply a fixed length on the random walk, we allow the generated vertex sequences have a variable length, in order to have a close analogy to the variable-length sentences in natural languages.

Generally speaking, the above generation process follows the principle of "rich gets richer", which is a physical phenomena existing in many real networks, i.e., the vertex connectivities follow a scale-free power-law distribution [35].

The workflow of our random walk generator is summarized in Algorithm 1, where $maxT$ and $minT$ are the maximal and minimal numbers of random walks starting from each vertex, respectively. D^U (or D^V) output by Algorithm 1 is the corpus generated from the vertex set U (or V). The vertex centrality can be measured by many metrics, such as degree centrality, PageRank and HITS [29], etc., and we use HITS in our experiments.

Algorithm 1: WalkGenerator(W, R, $maxT$, $minT$, p)

Input : weight matrix of the bipartite network $\mathbf{W}$, vertex set R (can be U or V), maximal walks per vertex $maxT$, minimal walks per vertex $minT$, walk stopping probability p

Output : a set of vertex sequences D^R

```
Calculate vertices' centrality: H = CentralityMeasure(W);
Calculate W^R w.r.t. Equation (4);
foreach vertex v_i ∈ R do
    l = max(H(v_i) × maxT, minT);
    for i = 0 to l do
        D_{v_i} = BiasedRamdomWalk(W^R, v_i, p);
        Add D_{v_i} into D^R;
return D^R;
```

4.2.2 ***Implicit Relation Modeling.*** After performing biased random walks on the two homogeneous networks respectively, we obtain two corpora of vertex sequences. Next we employ the Skip-gram model [11] on the two corpora to learn vertex embeddings.

The aim is to capture the high-order proximity, which assumes that vertices frequently co-occurred in the same context of a sequence should be assigned to similar embeddings. Given a vertex sequence S and a vertex u_i, the context is defined as the ws vertices before u_i and after u_i in S; each vertex is associated with a context vector $\vec{\boldsymbol{\theta}}_i$ (or $\vec{\boldsymbol{\vartheta}}_j$) to denote its role as a context. As there are two types of vertices in a bipartite network, we preserve the high-order proximities separately. Specifically, for the corpus D^U, the conditional probability to maximize is:

$$maximize\ O_2 = \prod_{u_i \in S \wedge S \in D^U} \prod_{u_c \in C_S(u_i)} P(u_c|u_i). \tag{5}$$

where $C_S(u_i)$ denotes the context vertices of vertex u_i in sequence S. Similarly, we can get the objective function for corpus D^V:

$$maximize\ O_3 = \prod_{v_j \in S \wedge S \in D^V} \prod_{v_c \in C_S(v_j)} P(v_c|v_j). \tag{6}$$

Following existing neural embedding methods [4, 8, 20], we parameterize the conditional probability $P(u_c|u_i)$ and $P(v_c|v_j)$ using the inner product kernel with softmax for output:

$$P(u_c|u_i) = \frac{\exp(\vec{\mathbf{u}}_i{}^T \vec{\boldsymbol{\theta}}_c)}{\sum_{k=1}^{|U|} \exp(\vec{\mathbf{u}}_i{}^T \vec{\boldsymbol{\theta}}_k)}, \quad P(v_c|v_j) = \frac{\exp(\vec{\mathbf{v}}_j{}^T \vec{\boldsymbol{\vartheta}}_c)}{\sum_{k=1}^{|V|} \exp(\vec{\mathbf{v}}_j{}^T \vec{\boldsymbol{\vartheta}}_k)}. \tag{7}$$

where $P(u_c|u_i)$ denotes how likely u_c is observed in the contexts of u_i; similar meaning applies to $P(v_c|v_j)$. With this definition, achieving the goal defined in Equations (5) and (6) will force the vertices with the similar contexts to be close in the embedding space. Nevertheless, optimizing the objectives is non-trivial, since each evaluation of the softmax function needs to traverse all vertices of a side, which is very time-costing. To reduce the learning complexity, we employ the idea of negative sampling [11].

4.2.3 ***Negative Sampling.*** The idea of negative sampling is to approximate the costly denominator term of softmax with some sampled negative instances [36]. Then the learning can be performed by optimizing a point-wise classification loss. For a center vertex u_i, high-quality negatives should be the vertices that are dissimilar from u_i. Towards this goal, some heuristics have been applied, such as sampling from popularity-biased non-uniform distribution [11]. Here we propose a more grounded sampling method that caters the network data.

First we employ locality sensitive hashing (LSH) [37] to block vertices after shingling each vertex by its ws-hop neighbors with respect to the topological structure in the input bipartite network. Given a center vertex, we then randomly choose the negative samples from the buckets that are different from the bucket contained the center vertex. Through this way, we can obtain high-quality and diverse negative samples, since LSH can guarantee that dissimilar vertices are located in different buckets in a probabilistic way [37].

Let $N_S^{ns}(u_i)$ denote the ns negative samples for a center vertex u_i in sequence $S \in D^U$, we can then approximate the conditional probability $p(u_c|u_i)$ defined in Equation (7) as:

$$p(u_c, N_S^{ns}(u_i)|u_i) = \prod_{z \in \{u_c\} \cup N_S^{ns}(u_i)} P(z|u_i), \tag{8}$$

where the probability $P(z|u_j)$ is defined as:

$$P(z|u_i) = \begin{cases} \sigma(\vec{\mathbf{u}}_i{}^T \vec{\boldsymbol{\theta}}_z), & \text{if } z \text{ is a context of } u_i \\ 1 - \sigma(\vec{\mathbf{u}}_i{}^T \vec{\boldsymbol{\theta}}_z), & z \in N_S^{ns}(u_i) \end{cases},$$

where σ denotes the sigmoid function $1/(1+e^{-x})$. By replacing $p(u_c|u_i)$ in Equation (5) with the definition of $p(u_c, N_S^{ns}(u_i)|u_i)$, we can get the approximated objective function to optimize. The semantics is that the proximity between the center vertex and their contextual vertices should be maximized, whereas the proximity between the center vertex and the negative samples should be minimized.

Following the similar formulations, we can get the counterparts for the conditional probability $p(v_c|v_j)$, the details of which are omitted here due to space limitation.

4.3 Joint Optimization

To embed a bipartite network by preserving both explicit and implicit relations simultaneously, we combine their objective functions to form a joint optimization framework.

$$maximize\ L = \alpha \log O_2 + \beta \log O_3 - \gamma O_1. \tag{9}$$

where parameters α, β and γ are hyper-parameters to be specified to combine different components in the joint optimization framework.

To optimize the joint model, we utilize the Stochastic Gradient Ascent algorithm (SGA). Note that the three components of Equation (9) have different definitions of a training instance. To handle this issue, we tweak the SGA algorithm by performing a gradient step as follows:

Step I: For a stochastic explicit relation, i.e., an edge $e_{ij} \in E$, we first update the embedding vectors $\vec{\mathbf{u}}_i$ and $\vec{\mathbf{v}}_j$ by utilizing SGA to maximize the last component $L_1 = -\gamma O_1$. We give the SGA update rule for $\vec{\mathbf{u}}_i$ and $\vec{\mathbf{v}}_j$ as follows:

$$\vec{\mathbf{u}}_i = \vec{\mathbf{u}}_i + \lambda\{\gamma w_{ij}[1 - \sigma(\vec{\mathbf{u}}_i{}^T \vec{\mathbf{v}}_j)] \cdot \vec{\mathbf{v}}_j\}, \tag{10}$$

$$\vec{\mathbf{v}}_j = \vec{\mathbf{v}}_j + \lambda\{\gamma w_{ij}[1 - \sigma(\vec{\mathbf{u}}_i{}^T \vec{\mathbf{v}}_j)] \cdot \vec{\mathbf{u}}_i\}, \tag{11}$$

where λ denotes the learning rate.

Step II: We then treat vertices u_i and v_j as the center vertex; by employing SGA to maximize objective functions $L_2 = \alpha \log O_2$ and $L_3 = \beta \log O_3$, we can preserve the implicit relations. Specifically, given the center vertex u_i (or v_j) and its context vertex u_c (or v_c), we update their embedding vectors $\vec{\mathbf{u}}_i$ (or $\vec{\mathbf{v}}_j$) as follows:

$$\vec{\mathbf{u}}_i = \vec{\mathbf{u}}_i + \lambda\{\sum_{z \in \{u_c\} \cup N_S^{ns}(u_i)} \alpha[I(z, u_i) - \sigma(\vec{\mathbf{u}}_i{}^T \vec{\boldsymbol{\theta}}_z)] \cdot \vec{\boldsymbol{\theta}}_z\} \tag{12}$$

$$\vec{\mathbf{v}}_j = \vec{\mathbf{v}}_j + \lambda\{\sum_{z \in \{v_c\} \cup N_S^{ns}(v_j)} \beta[I(z, v_j) - \sigma(\vec{\mathbf{v}}_j{}^T \vec{\boldsymbol{\vartheta}}_z)] \cdot \vec{\boldsymbol{\vartheta}}_z\} \tag{13}$$

where $I(z, u_i)$ is an indicator function that determines whether vertex z is in the context of u_i or not; similar meaning applies to $I(z, v_j)$. Furthermore, the context vectors of both positive and negative instances are updated as:

$$\vec{\boldsymbol{\theta}}_z = \vec{\boldsymbol{\theta}}_z + \lambda\{\alpha[I(z, u_i) - \sigma(\vec{\mathbf{u}}_i{}^T \vec{\boldsymbol{\theta}}_z)] \cdot \vec{\mathbf{u}}_i\}, \tag{14}$$

$$\vec{\boldsymbol{\vartheta}}_z = \vec{\boldsymbol{\vartheta}}_z + \lambda\{\beta[I(z, v_j) - \sigma(\vec{\mathbf{v}}_j{}^T \vec{\boldsymbol{\vartheta}}_z)] \cdot \vec{\mathbf{v}}_j\}. \tag{15}$$

We summarize the learning process in Algorithm 2. To be specific, lines 1-2 initialize all embedding vectors and context vectors; lines 3-4 generate the corpus of vertex sequences; lines 8 and 12 perform negative sampling; lines 9-10 and 13-14 employ SGA to learn the embeddings.

Algorithm 2: Training algorithm of BiNE

Input : bipartite network $G = (U, V, E)$, weight matrix of the bipartite network $\mathbf{W}$, window size ws, number of negative samples ns, embedding size d, maximal walks per vertex $maxT$, minimal walks per vertex $minT$, walk stopping probability p

Output : vertex embeding matrices $\mathbf{U}$ and $\mathbf{V}$

Initialize embedding vectors $\vec{\mathbf{u}}_i$ and $\vec{\mathbf{v}}_j$;
Initialize context vectors $\vec{\boldsymbol{\theta}}_i$ and $\vec{\boldsymbol{\vartheta}}_j$;
D^U = WalkGenerator($\mathbf{W}, U, maxT, minT, p$);
D^V = WalkGenerator($\mathbf{W}, V, maxT, minT, p$);
foreach *edge* $(u_i, v_j) \in E$ **do**
  Update $\vec{\mathbf{u}}_i$ and $\vec{\mathbf{v}}_j$ using Equations (10) and (11);
  foreach (u_i, u_c) *in the sequence* $S \in D^U$ **do**
    Negative sampling to generate $N_S^{ns}(u_i)$;
    Update $\vec{\mathbf{u}}_i$ using Equation (12);
    Update $\vec{\boldsymbol{\theta}}_z$ using Equation (14) where $z \in \{u_c\} \cup N_S^{ns}(u_i)$;
  foreach (v_j, v_c) *in the sequence* $S \in D^V$ **do**
    Negative sampling to generate $N_S^{ns}(v_j)$;
    Update $\vec{\mathbf{v}}_j$ using Equation (13);
    Update $\vec{\boldsymbol{\vartheta}}_z$ using Equation (15) where $z \in \{v_c\} \cup N_S^{ns}(v_j)$;
return Vertex embeding matrices $\mathbf{U}$ and $\mathbf{V}$

4.4 Discussions

Pre-training. The joint objective function of BiNE in Equation (9) is non-convex, so initialization plays an important role to find a good solution. We pre-train Equation (3) to get the initial vertex embeddings.

Computational Complexity Analysis. The corpus generation and joint model optimization are two key processes of BiNE. However, the complexity of generating corpus will be increased if $\mathbf{W}^U$ or $\mathbf{W}^V$ becomes dense. To avoid processing the dense matrix, an alternative way is to walk two steps in the original bipartite network. Suppose that vc is the visitation count of vertex v in the generated corpus. The context size is therefore $vc \cdot 2ws$. It may be a big value for vertices having high degrees, yet we only randomly select a small batch of the contextual vertices, e.g., bs ($bs \ll vc$). Thus, the complexity of algorithm is $O(2|E| \cdot bs \cdot 2ws \cdot (ns + 1))$, where ns is the number of negative samples. To some extent, all the contextual vertices of a center vertex can be trained in each iteration by setting a proper bs, because the center vertex will be visited more than once when traversing all edges.

5 EXPERIMENTS

To evaluate the vertex embeddings learned by BiNE, we employ them to address two representative applications of bipartite network mining — link prediction and recommendation. Link prediction is usually approached as a classification task that predicts whether a link exists between two vertices, and recommendation is a personalized ranking task that aims to provide items of interest for a user. Through empirical evaluation, we aim to answer the following research questions:

RQ1 How does BiNE perform compared with state-of-the-art network embedding methods and other representative baselines of the two applications?
RQ2 Is the modeling of implicit relations helpful to learn more desirable representations for bipartite networks?
RQ3 Can our proposed random walk generator contribute to learning better vertex representations?
RQ4 How do the key hyper-parameters affect the performance of BiNE?

In what follows, we first introduce the experimental settings, and then answer the above research questions in turn. Furthermore, we perform a case study, which visualizes a small bipartite network, to demonstrate the rationality of BiNE.

5.1 Experimental Settings

5.1.1 ***Dataset***. (1) For the link prediction task, we use two unweighted bipartite networks constructed from Wikepedia and Tencent, respectively. Specifically, the Wikipedia dataset is publicly accessible[2], which contains the edit relationship between authors and pages; the Tencent dataset records the watching behaviors of users on movies in QQlive[3] in one month's time. (2) For the recommendation task, we use three weighted bipartite networks constructed from DBLP, MovieLens, and VisualizeUs. Specifically, the DBLP dataset[4] contains the publish network of authors on venues, where the edge weight indicates the number of papers published on a venue by an author. The MovieLens dataset [5] has been widely used for evaluating movie recommender systems [33], where the edge weight denotes the rating score of a user on an item. The VisualizeUS dataset[6] contains the picture tagging network between pictures and tags, where the edge weight denotes the number of times a tag has been tagged on an image. The statistics of our experimented datasets are summarized in Table 1.

Note that our experimented datasets cover a wide range of applications based on bipartite networks, which can test the universality of our BiNE method. Moreover, we have purposefully chosen weighted networks for the recommendation task, so as to study BiNE's ability in embedding weighted bipartite networks.

Table 1: Statistics of bipartite networks and metrics adopted in experiments for different tasks.

Task	Link Prediction		Recommendation		
Type	undirected, unweighted		undirected, weighted		
Metric	AUC-ROC,AUC-PR		F1, NDCG, MAP, MRR		
Name	Tencent	Wikipedia	VisualizeUs	DBLP	MovieLens
$\|U\|$	14,259	15,000	6,000	6,001	69,878
$\|V\|$	1,149	3,214	3,355	1,308	10,677
$\|E\|$	196,290	172,426	35,639	29,256	10,000,054
Density	1.2%	0.4%	0.2%	0.4%	1.3%

[2] http://konect.uni-koblenz.de/networks/wikipedia_link_en
[3] https://v.qq.com/
[4] http://dblp.uni-trier.de/xml/
[5] http://grouplens.org/datasets/movielens/
[6] http://konect.uni-koblenz.de/networks/pics_ti

5.1.2 ***Evaluation Protocols.*** (1) To evaluate the link prediction task, we apply the same protocol as the Node2vec paper [4]. Specifically, for the Wikepedia dataset, the observed links are treated as positive instances, and we randomly sample an equal number of vertex pairs that are not connected as the negative instances. For the Tencent dataset, positive instances include user-movie pairs where the user has watched the movie for more than 5 minutes, otherwise, it is treated as an negative instance. For both datasets, we randomly sample 60% instances as the training set, evaluating performance on the remaining 40% of testing set. Following the previous work [38], we employ the ROC curve (AUC-ROC) and Precison-Recall curve (AUC-PR) to evaluate the link prediction performance. (2) For the recommendation task, we adopt the same setting for all the three datasets. We randomly sample 60% edges as the training data, using the remaining 40% edges as the ground-truth for testing. For each user, we rank all items in her testing set and evaluate the ranking list with four IR metrics: F1, Normalized Discounted Cumulative Gain (NDCG), Mean Average Precision (MAP), and Mean Reciprocal Rank (MRR). We truncate the ranking list at 10 to study the performance of top-10 recommendation. For each metric, we compute the average score for all users, and perform paired sample T-test on it. To avoid over-fitting, we generate 10 folds of train-test split, tuning hyper-parameters on the first fold only for each method. We use the optimal hyper-parameter setting and report the average performance of all folds (i.e., the score of each metric and the p-value of t-test).

5.1.3 ***Baselines.*** We compare BiNE with three types of baselines:

(1) **Network Embedding Methods**. Similar to BiNE, this set of methods also learn vertex embeddings and are representative of state-of-the-art network embedding methods. For each method, we use the released implementations for our experiments.
- DeepWalk [8]: As a homogeneous network embedding method, DeepWalk performs uniform random walks to get a corpus of vertex sequences. Then the word2vec is applied on the corpus to learn vertex embeddings.
- LINE [20]: This approach optimizes both the 1st-order and 2nd-order proximities in a homogeneous network. We use the LINE(1st+2nd) method which has shown the best results in their paper.
- Node2vec [4]: This method extends DeepWalk by performing biased random walks to generate the corpus of vertex sequences. The hyper-parameters p and q are set to 0.5 which has empirically shown good results.
- Metapath2vec++ [14]: This is the state-of-the-art method for embedding heterogeneous networks.The meta-path scheme chosen in our experiments are "IUI" (item-user-item) and "IUI"+"UIU" (user-item-user), and we only report the best result between them.

(2) To benchmark the link prediction task, we also compare with a set of methods that are specifically designed for the task. We apply several indices proposed in [38], including Jaccard Coefficient (JC), Adamaic/Adar (AA), Katz Index (Katz), and Preferential Attachmenthave (PA).

(3) We compare with several competitive methods[7] that are designed for the top-K item recommendation task.
- BPR [31]:This method optimizes the matrix factorization (MF) model with a pairwise ranking-aware objective. This method has been widely used in recommendation literature as a highly competitive baseline [33].
- RankALS [39]: This method also optimizes the MF model for the ranking task, by towards a different pairwise regression-based loss.
- FISMauc [40]: Distinct to MF, factored item similarity model (FISM) is an item-based collaborative filtering method. We employ the AUC-based objective to optimize FISM for the top-K task.

Table 2: The search range and optimal setting (highlighted in red) of hyper-parameters for our BiNE method.

Parameter	Meaning	Test values
ns	number of negative samples	[1, 2, **4**, 6, 8, 10]
ws	size of window	[1, 3, **5**, 7, 9]
p	walk stopping probability	[0.05, 0.1, **0.15**, 0.2, 0.3, 0.4, 0.5]
β	trade-off parameter	[0.0001, 0.001, **0.01**, 0.1, 1]
γ	trade-off parameter	[0.01, 0.05, **0.1**, 0.5, **1**, 5]

5.1.4 ***Parameter Settings.*** We have fairly tuned the hyper-parameters for each method. For all network embedding methods, we set the embedding size as 128 for a fair comparison. For the recommendation baselines, we tuned the learning rate and latent factor number since they impact most on the performance; other hyper-parameters follow the default setting of the LibRec toolkit. For our BiNE, we fix the loss trade-off parameter α as 0.01 and tune the other two. The $minT$ and $maxT$ are respectively set to 1 and 32, which empirically show good results. We test the learning rate λ of [0.01, 0.025, 0.1]. And the optimal setting of learning rate is 0.025 for the VisualizeUs/DBLP dataset and 0.01 for others. The search range and optimal setting (highlighted in red font) of other parameters are shown in Table 2. Note that besides γ is set differently — 0.1 for recommendation and 1 for link prediction — other parameters are set to the same value for both tasks.

5.2 Performance Comparison (RQ1)

Table 3: Link prediction performance on Tencent and Wikipedia.

Algorithm	Tencent		Wikipedia	
	AUC-ROC	AUC-PR	AUC-ROC	AUC-PR
JC	51.49%	66.18%	63.90%	73.04%
AA	50.63%	65.66%	87.37%	91.12%
Katz	50.90%	65.06%	90.84%	92.42%
PA	55.60%	68.99%	90.71%	93.37%
DeepWalk	57.62%	71.32%	89.71%	91.20%
LINE	59.68%	73.48%	91.62%	93.28%
Node2vec	59.28%	72.62%	89.93%	91.23%
Metapath2vec++	60.70%	73.69%	89.56%	91.72%
BiNE	**60.98%****	**73.77%****	**92.91%****	**94.45%****

** indicates that the improvements are statistically significant for $p < 0.01$ judged by paired t-test.

[7] We use the implementations from LibRec: https://www.librec.net/

Table 4: Performance comparison of Top-10 Recommendation on VisualizeUs, DBLP, and MovieLens.

Algorithm	VisualizeUs				DBLP				Movielens			
	F1@10	NDCG@10	MAP@10	MRR@10	F1@10	NDCG@10	MAP@10	MRR@10	F1@10	NDCG@10	MAP@10	MRR@10
BPR	6.22%	9.52%	5.51%	13.71%	8.95%	18.38%	13.55%	22.25%	8.03%	7.58%	2.23%	40.81%
RankALS	2.72%	3.29%	1.50%	3.81%	7.62%	11.50%	7.52%	14.87%	8.48%	7.95%	2.66%	38.93%
FISMauc	10.25%	15.46%	8.86%	16.67%	9.81%	13.77%	7.38%	14.51%	6.77%	6.13%	1.63%	34.04%
DeepWalk	5.82%	8.83%	4.28%	12.12%	8.50%	24.14%	19.71%	31.53%	3.73%	3.21%	0.90%	15.40%
LINE	9.62%	13.76%	7.81%	14.99%	8.99%	14.41%	9.62%	17.13%	6.91%	6.50%	1.74%	38.12%
Node2vec	6.73%	9.71%	6.25%	13.95%	8.54%	23.89%	19.44%	31.11%	4.16%	3.68%	1.05%	18.33%
Metapath2vec++	5.92%	8.96%	5.35%	13.54%	8.65%	25.14%	19.06%	31.97%	4.65%	4.39%	1.91%	16.60%
BiNE	**13.63%****	**24.50%****	**16.46%****	**34.23%****	**11.37%****	**26.19%****	**20.47%****	**33.36%****	**9.14%****	**9.02%****	**3.01%****	**45.95%****

** indicates that the improvements are statistically significant for $p < 0.01$ judged by paired t-test.

5.2.1 ***Link Prediction.*** In this task, embedding vectors given by BiNE are treated as feature vectors of a logistic regression classifier. Specifically, given a vertex pair (u_i, v_j), we feed their embedding vectors $\vec{\mathbf{u}}_i$ and $\vec{\mathbf{v}}_j$ into the classifier, which is trained on observed links of the bipartite network. Table 3 illustrates the performance of baselines and our BiNE, where we have the following key observations:

- The neural network-based methods outperform the indices proposed in [38] significantly. This is due to the factors that: (1) one index proposed in [38] only emphasizes one kind of network topological structure, rather than the global structure; (2) the neural network-based methods predict the links in a data-dependent supervised manner, which is more advantageous.
- Metapath2vec++ and BiNE are significantly better than other neural network-based methods. This points to the positive effect of modeling both explicit and implicit relations into the embedding process.
- BiNE outperforms Metapath2vec++ significantly and achieves the best performance on both datasets in both metrics. This improvement demonstrates the effectiveness of our modeling of explicit and implicit relations in different ways, whereas Metapath2vec++ simply treats them as contributing equally to the learning.

5.2.2 ***Recommendation.*** We adopt the inner product kernel $\vec{\mathbf{u}}_i^T \vec{\mathbf{v}}_j$ to estimate the preference of user u_i on item v_j, and evaluate performance on the top-ranked results. Table 4 shows the performance of baselines and our BiNE, where we have the following key observations:

- BiNE outperforms all baselines on all datasets, and the improvements are more significant on VisualizeUs — the most sparse dataset among the three. This sheds lights on the benefit of preserving both explicit and implicit relations in a bipartite network. Although Metapath2vec++ also preserves both explicit and implicit relations, we did not observe consistently good results since it ignores the weights and treats the two types of relations as equally.
- BiNE outperforms LINE significantly. The suboptimal performance of LINE are twofold: (1) although LINE preserves both 1st-order and 2nd-order relations to learn network embeddings, it ignores further higher-order proximities among vertices; (2) LINE learns two seperated embeddings for 1st-order and 2nd-order relations and concatenates them via post-processing, rather than optimizing them in a unified framework. As such, it reveals that: (1) only 2nd-order relations are insufficient for learning vertex embeddings for a bipartite network; (2) it is necessary to build a joint model to capture both explicit and implicit relations. These are the two featured designs of our BiNE.

5.3 Utility of Implicit Relations (RQ2)

Table 5: BiNE with and without implicit relations.

	Without Implicit Relations		With Implicit Relations	
Link Prediction				
Dataset	**AUC-ROC**	**AUC-PR**	**AUC-ROC**	**AUC-PR**
Tencent	59.78%	73.05%	**60.98%****	**73.77%****
WikiPedia	91.47%	93.73%	**92.91%****	**94.45%****
Recommendation				
Dataset	**MAP@10**	**MRR@10**	**MAP@10**	**MRR@10**
VisualizeUS	7.91%	15.65%	**16.46%****	**34.23%****
DBLP	20.20%	32.95%	**20.47%****	**33.36%****
MovieLens	2.86%	43.98%	**3.01%****	**45.95%****

** indicates that the improvements are statistically significant for $p < 0.01$ judged by paired t-test.

To demonstrate the effectiveness of integrating explicit and implicit relations, we compare BiNE with its variant that removes the modeling of implicit relations. We only show the performance on MAP@10 and MRR@10 on recommendation due to space limitation. From Table 5, we can find that the largest absolute improvements of BiNE with implicit relations are 1.44% and 18.58% for link prediction and recommendation, respectively. It indicates that our proposed way of modeling high-order implicit relations is rather effective to complement with explicit relation modeling.

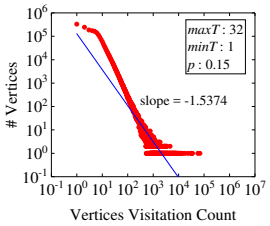


Figure 3: Distribution of vertices in the biased and self-adpative random walk generator.

5.4 Random Walk Generator (RQ3)

We design a new random walk generator for BiNE to preserve properties of bipartite networks in the generated corpora as much

Table 6: BiNE with different random walk generators.

	Uniform Random Walk Generator		Biased and Self-adaptive Random Walk Generator	
Link Prediction				
Dataset	**AUC-ROC**	**AUC-PR**	**AUC-ROC**	**AUC-PR**
Tencent	59.75%	73.06%	**60.98%****	**73.77%****
WikiPedia	88.77%	91.91%	**92.91%****	**94.45%****
Recommendation				
Dataset	**MAP@10**	**MRR@10**	**MAP@10**	**MRR@10**
VisualizeUS	15.93%	33.66%	**16.46%****	**34.23%****
DBLP	11.79%	23.41%	**20.47%****	**33.66%****
MovieLens	2.91%	46.12%	**3.04%****	**46.20%****

** indicates that the improvements are statistically significant for $p < 0.01$ judged by paired t-test.

as possible, such as the importance and distribution of vertices. To guarantee comparability with Figure 1, we also use the YouTube's video network as the input of our random walk generator and show distribution of vertices. As shown in Figure 3, our random walk generator almost generates a standard power-law distribution with a slope -1.537 which is very close to that of the original network (-1.58186).

We also compare the performance of BiNE under two settings — use or not use our proposed random walk generator. As shown in Table 6, the biggest absolute improvements of BiNE using our proposed random walk generator are 4.14% and 10.25% for link prediction and recommendation, respectively. It indicates that the biased and self-adaptive random walk generator contributes to improving the vertex embeddings.

Note that we change the default value of $maxT$ to 128 for this empirical study on Movielens dataset. This is due to the factor that Movielens is the biggest and densest bipartite network data compared to the others. The default value of $maxT$ may be too small to fully preserve the implicit relations. This indicates that $maxT$ should be specified to a large number for a large-scale network.

5.5 Hyper-parameter Studies (RQ4)

Due to space limitation, we only investigate the impact of the hyper parameters β, γ (n.b., we fix $\alpha = 0.01$) since parameters β, and γ play crucial roles to balance the impacts of the explicit relations (γ) and implicit relations (α, β) for vertex embeddings. We analyze both link prediction and recommendation tasks on datasets VisualizeUs and Wikipedia, respectively. Except for the parameters being tested, other parameters assume default values.

From Figure 4 and Figure 5, we observe that the impact of the two parameters have similar trends w.r.t. different performance metrics in the same task: (1) with the increase of γ, the performance first increases and then remains stable after certain values; (2) with the increase of β, the performance first increases and then decreases after certain values. When γ is small, our optimization model may artificially reduce the importance of the explicit relations for network embeddings. However, when β is large, the optimization model may artificially overstate the role of the implicit relations.

The existence of the yielding points confirms that purely using explicit or implicit relations is insufficient to learn desirable representations for a bipartite network. In addition, we can observe that the best value of γ is larger than that of α and β. It indicates that the explicit relations are more important than implicit relations for the bipartite network embeddings.

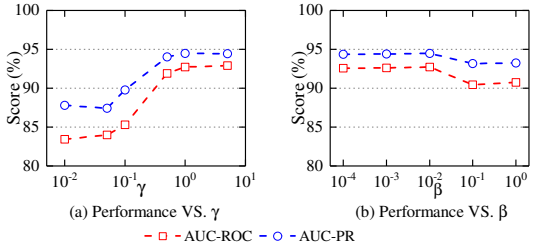


Figure 4: Impact of hyper-parameters on link prediction.

5.6 Case Study

A good embedding algorithm should provide a meaningful visualization that layout a network. Due to space limitation, we conduct a visualization study for a small bipartite network. We visualize a collaboration bipartite network, which is a subset of DBLP dataset. It consists of 736 researchers and 6 international journals, where the 6 journals are from two different research fields: SICOMP, IANDC and TIT from computer science theory, and AI, IJCV,and JMLR from artificial intelligence. A link will be formed if the researcher published at least 5 papers in the journal. The research field of a researcher is determined by the published venues of his/her works.

We utilize the t-SNE tool [41] to map the embedding vectors of authors into 2D space. Figure 6 compares the visualization results given by different embedding approaches, where color of a vertex indicates the research field of a researcher (red: "computer science theory", blue: "artificial intelligence"). Obviously, DeepWalk, Node2vec, LINE, Metapath2vec++, and our BiNE are good since researchers belonging different research fields are well seperated.

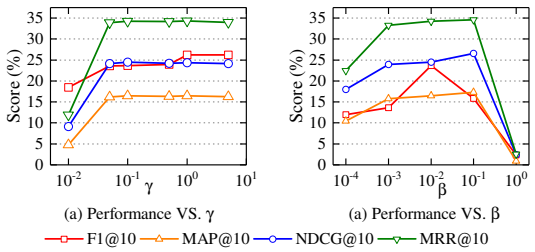


Figure 5: Impact of hyper-parameters on recommendation.

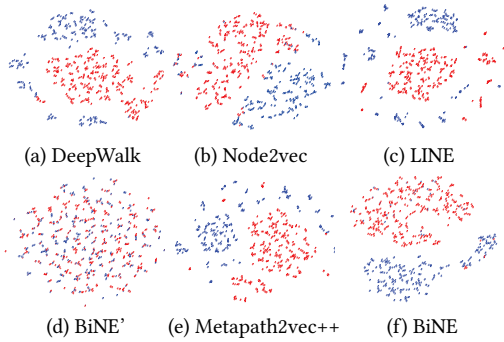


Figure 6: Visualization of authors in DBLP. Color of a vertex indicates the research fields of the authors (red: "computer science theory", blue: "artificial intelligence"). BiNE' is the version of BiNE – without implicit relations.

In our opinion, BiNE gives a better result due to the fact that it also generates an obvious gap between two research fields. However, BiNE' (a variant of BiNE – without implicit relations) demonstates a worse layout than expected. It indicates that modeling high-order implicit relations is helpful to preserve the network structure well.

6 CONCLUSIONS

We have presented BiNE, a novel approach for embedding bipartite networks. It jointly models both the explicit relations and high-order implicit relations in learning the representation for vertices. Extensive experiments on several tasks of link prediction, recommendation, and visualization demonstrate the effectiveness and rationality of our BiNE method.

In this work, we have only considered the information revealed in observed edges, thus it may fail for vertices that have few or even no edges. Since missing data is a common situation in real-world applications, the observed edges may not contain sufficient signal on vertex relations. To address this issue, we plan to extend our BiNE method to model auxiliary side information, such as numerical features [42], textual descriptions [43], and among other attributes [9]. In addition, the bipartite networks in many practical applications are dynamically updated [32]. Thus, we plan to investigate how to efficiently refresh embeddings for dynamic bipartite networks.

7 ACKNOWLEDGMENTS

This work has been supported by the National Key Research and Development Program of China under grant 2016YFB1000905, and the National Natural Science Foundation of China under Grant No. U1401256, 61672234, 61502236, and 61472321. NExT research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its IRC@SG Funding Initiative , and the Shanghai Agriculture Applied Technology Development Program, China (Grant No.T20170303).

REFERENCES

[1] Hongbo Deng, Michael R. Lyu, and Irwin King. A generalized co-hits algorithm and its application to bipartite graphs. In *KDD*, pages 239–248, 2009.
[2] Shan Jiang, Yuening Hu, Changsung Kang, Tim Daly Jr., Dawei Yin, Yi Chang, and ChengXiang Zhai. Learning query and document relevance from a web-scale click graph. In *SIGIR*, pages 185–194, 2016.
[3] Xiangnan He, Ming Gao, Min-Yen Kan, and Dingxian Wang. Birank: Towards ranking on bipartite graphs. *TKDE*, 29(1):57–71, 2017.
[4] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, pages 855–864, 2016.
[5] Qingyao Ai, Yongfeng Zhang, Keping Bi, Xu Chen, and W. Bruce Croft. Learning a hierarchical embedding model for personalized product search. In *SIGIR*, pages 645–654, 2017.
[6] Xiangnan He and Tat-Seng Chua. Neural factorization machines for sparse predictive analytics. In *SIGIR*, pages 355–364, 2017.
[7] Da Cao, Liqiang Nie, Xiangnan He, Xiaochi Wei, Shunzhi Zhu, and Tat-Seng Chua. Embedding factorization models for jointly recommending items and user generated lists. In *SIGIR*, pages 585–594, 2017.
[8] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: online learning of social representations. In *KDD*, pages 701–710, 2014.
[9] Lizi Liao, Xiangnan He, Hanwang Zhang, and Tat-Seng Chua. Attributed social network embedding. *TKDE*, 2018.
[10] Jundong Li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. Attributed network embedding for learning in a dynamic environment. In *CIKM*, pages 387–396, 2017.
[11] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013.
[12] Lu Yu, Chuxu Zhang, Shichao Pei, Guolei Sun, and Xiangliang Zhang and. Walkranker: A unified pairwise ranking model with multiple relations for item recommendation. In *AAAI*, 2018.
[13] Suchit Pongnumkul and Kazuyuki Motohashi. A bipartite fitness model for online music streaming services. *Physica A: Statistical Mechanics and its Applications*, 490:1125–1137, 2018.
[14] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. metapath2vec: Scalable representation learning for heterogeneous networks. In *KDD*, pages 135–144. ACM, 2017.
[15] Meng Jiang, Peng Cui, Nicholas Jing Yuan, Xing Xie, and Shiqiang Yang. Little is much: Bridging cross-platform behaviors through overlapped crowds. In *AAAI*, pages 13–19, 2016.
[16] Trevor F Cox and Michael AA Cox. *Multidimensional scaling*. CRC press, 2000.
[17] Angelia Nedic and Asuman E. Ozdaglar. A geometric framework for nonconvex optimization duality using augmented lagrangian functions. *J. Global Optimization*, 40(4):545–573, 2008.
[18] Meng Wang, Weijie Fu, Shijie Hao, Hengchang Liu, and Xindong Wu. Learning on big graph: Label inference and regularization with anchor hierarchy. *TKDE*, 29(5):1101–1114, 2017.
[19] Meng Wang, Weijie Fu, Shijie Hao, Dacheng Tao, and Xindong Wu. Scalable semi-supervised learning by efficient anchor graph regularization. *TKDE*, 28(7):1864–1877, 2016.
[20] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. LINE: large-scale information network embedding. In *WWW*, pages 1067–1077, 2015.
[21] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *KDD*, pages 1225–1234, 2016.
[22] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *CIKM*, pages 891–900, 2015.
[23] Xiao Huang, Jundong Li, and Xia Hu. Label informed attributed network embedding. In *WSDM*, pages 550–558, 2017.
[24] Jifan Chen, Qi Zhang, and Xuanjing Huang. Incorporate group information to enhance network embedding. In *CIKM*, pages 1901–1904, 2016.
[25] Chuan-Ju Wang, Ting-Hsiang Wang, Hsiu-Wei Yang, Bo-Sin Chang, and Ming-Feng Tsai. Ice: Item concept embedding via textual information. In *SIGIR*, pages 85–94, 2017.
[26] Min Xie, Hongzhi Yin, Hao Wang, Fanjiang Xu, Weitong Chen, and Sen Wang. Learning graph-based POI embedding for location-based recommendation. In *CIKM*, pages 15–24, 2016.
[27] Shiyu Chang, Wei Han, Jiliang Tang, Guo-Jun Qi, Charu C. Aggarwal, and Thomas S. Huang. Heterogeneous network embedding via deep architectures. In *KDD*, pages 119–128, 2015.
[28] Linchuan Xu, Xiaokai Wei, Jiannong Cao, and Philip S. Yu. Embedding of embedding (EOE): joint embedding for coupled heterogeneous networks. In *WSDM*, pages 741–749, 2017.
[29] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. In *ACM-SIAM*, pages 668–677, 1998.
[30] Hanwang Zhang, Fumin Shen, Wei Liu, Xiangnan He, Huanbo Luan, and Tat-Seng Chua. Discrete collaborative filtering. In *SIGIR*, pages 325–334, 2016.
[31] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. BPR: bayesian personalized ranking from implicit feedback. In *UAI*, pages 452–461, 2009.
[32] Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. Fast matrix factorization for online recommendation with implicit feedback. In *SIGIR*, pages 549–558, 2016.
[33] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *WWW*, pages 173–182, 2017.
[34] Taher Alzahrani, Kathy J. Horadam, and Serdar Boztas. Community detection in bipartite networks using random walks. In *CompleNet*, pages 157–165, 2014.
[35] Jure Leskovec, Deepayan Chakrabarti, Jon M. Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. Kronecker graphs: An approach to modeling networks. *JMLR*, 11:985–1042, 2010.
[36] Hongzhi Yin, Lei Zou, Quoc Viet Hung Nguyen, Zi Huang, and Xiaofang Zhou. Joint event-partner recommendation in event-based social networks. In *ICDE*, 2018.
[37] Hongya Wang, Jiao Cao, LihChyun Shu, and Davood Rafiei. Locality sensitive hashing revisited: filling the gap between theory and algorithm analysis. In *CIKM*, pages 1969–1978, 2013.
[38] Shuang Xia, Bing Tian Dai, Ee-Peng Lim, Yong Zhang, and Chunxiao Xing. Link prediction for bipartite social networks: The role of structural holes. In *ASONAM*, pages 153–157, 2012.
[39] Gábor Takács and Domonkos Tikk. Alternating least squares for personalized ranking. In *RecSys*, pages 83–90, 2012.
[40] Santosh Kabbur, Xia Ning, and George Karypis. FISM: factored item similarity models for top-n recommender systems. In *KDD*, pages 659–667, 2013.
[41] Van Der Maaten Laurens, Geoffrey Hinton, and Geoffrey Hinton, Van Der Maaten. Visualizing data using t-sne. *JMLR*, 9(2605):2579–2605, 2008.
[42] Fuli Feng, Xiangnan He, Yiqun Liu, Liqiang Nie, and Tat-Seng Chua. Learning on partial-order hypergraphs. In *WWW*, pages 1523–1532, 2018.
[43] Tuan M. V. Le and Hady Wirawan Lauw. Probabilistic latent document network embedding. In *ICDM*, pages 270–279, 2014.