Bringing Order to Network Embedding: A Relative Ranking based Approach

Yaojing Wang[†], Guosheng Pan[†], Yuan Yao[†], Hanghang Tong[‡], Hongxia Yang[§], Feng Xu[†], Jian Lu[†]
[†] State Key Laboratory for Novel Software Technology, Nanjing University, China
[‡] University of Illinois at Urbana-Champaign, USA
[§] Alibaba Group, China
{wyj,pgs}@smail.nju.edu.cn,{y.yao,xf,lj}@nju.edu.cn,htong@illinois.edu,yang.yhx@alibaba-inc.com

ABSTRACT

Network embedding aims to automatically learn the node representations in networks. The basic idea of network embedding is to first construct a network to describe the neighborhood context for each node, and then learn the node representations by designing an objective function to preserve certain properties of the constructed context network. The vast majority of the existing methods, explicitly or implicitly, follow a *pointwise* design principle. That is, the objective can be decomposed into the summation of the certain goodness function over each *individual edge* of the context network. In this paper, we propose to go beyond such pointwise approaches, and introduce the ranking-oriented design principle for network embedding. The key idea is to decompose the overall objective function into the summation of a goodness function over a *set of edges* to collectively preserve their relative rankings on the context network. We instantiate the ranking-oriented design principle by two new network embedding algorithms, including a *pairwise* network embedding method PaWine which optimizes the relative weights of *edge pairs*, and a listwise method LiWine which optimizes the relative weights of *edge lists*. Both proposed algorithms bear a linear time complexity, making themselves scalable to large networks. We conduct extensive experimental evaluations on five real datasets with a variety of downstream learning tasks, which demonstrate that the proposed approaches consistently outperform the existing methods.

CCS CONCEPTS

• **Information systems** → **Data mining**; • **Computing methodologies** → *Knowledge representation and reasoning*.

KEYWORDS

Network embedding, pairwise ranking, listwise ranking

ACM Reference Format:
Yaojing Wang[†], Guosheng Pan[†], Yuan Yao[†], Hanghang Tong[‡], Hongxia Yang[§], Feng Xu[†], Jian Lu[†]. 2020. Bringing Order to Network Embedding: A Relative Ranking based Approach. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3340531.3412041

CIKM '20, October 19–23, 2020, Virtual Event, Ireland

ACM ISBN 978-1-4503-6859-9/20/10…$15.00
https://doi.org/10.1145/3340531.3412041

1 INTRODUCTION

Network embedding, which aims to automatically learn the node representations/embeddings in networks, has been attracting much research interest, largely due to its strong empirical performance in many network analysis tasks including node classification [40], node clustering [57], and link prediction [19].

To date, many network embedding methods have been proposed. Generally speaking, a typical network embedding algorithm consists of two major steps: (1) construct the *context network* for each node based on the original network (Step 1), and (2) learn the node embeddings by designing an *objective function* which preserves certain properties of the constructed context matrix (Step 2). State-of-the-art, while rich in various ways to construct the context network (Step 1), almost exclusively follows a *pointwise* design principle for Step 2, whose objective function can be decomposed into the summation of the certain separate goodness function over each *individual edge* of the context network. The intuition of such pointwise methods is that the learned embeddings should match each (positive or negative) edge in the context network. However, other important semantics, e.g., relative rankings of a set of edges, might be overlooked.

In this paper, we first conduct a systematic analysis of the existing network embedding methods based on two orthogonal dimensions, i.e., context network construction (Step 1) and objective function design (Step 2). Based on this analysis, we propose to introduce *ranking-oriented* design principle into network embedding, whose key idea is to decompose the objective function into the summation of a goodness function over a *set of edges* to collectively preserve their relative rankings. The ranking-oriented design opens the door to a whole family of new network embedding methods - it can either (1) be used in conjunction with or (2) act as a substitute of the current pointwise objective functions. For the former, we propose PaWine which augments the existing pointwise objective function with a *pairwise* objective function. The intuition of PaWine is that the learned embeddings should not only match the observed edges, but also characterize the relative weights of edge pairs. For the latter, we propose a *listwise* objective function LiWine to characterize the relative orders of edge lists (i.e., the relative weights of multiple edges). Moreover, we develop efficient algorithms to solve both proposed objectives with a linear time complexity.

We conduct experiments by utilizing the learned embeddings for a variety of downstream network mining tasks (e.g., multi-label prediction, clustering, and multi-class prediction) to validate the effectiveness of the proposed approaches. The results reveal interesting interaction patterns between context network construction and objective design, and show that the proposed approaches consistently outperform the existing methods in terms of the prediction accuracy. For example, the proposed methods achieve better results even when directly applied on the original network, i.e., using the input network as the context network. Thus, it helps save the computational efforts to use short random walk or its variants to build a much denser context network as required by many existing methods. Our evaluations also indicate that the strength of the proposed PaWine and LiWine are complementary with each other. That is, the pairwise objective term in PaWine is more effective when integrated with the pointwise objective functions during the learning process; whereas LiWine is more effective when acting as a fine-tuned post-processing step of the existing pointwise embedding results.

In summary, the main contributions of this paper include:

- **Design Principle**. Based on a systematic analysis of state-of-the-art, we introduce a new ranking-oriented design principle for network embedding for characterizing the relative rankings of a set of edges collectively.
- **Algorithms and Analysis**. We propose two ranking-oriented network embedding approaches PaWine and LiWine. PaWine adopts a pairwise loss function in conjunction with the existing pointwise methods; LiWine encodes a listwise loss function. Both algorithms bear a linear time complexity.
- **Experimental Evaluations**. We conduct extensive experimental evaluations on five benchmark datasets demonstrating the effectiveness of the proposed methods. For example, the proposed PaWine algorithm achieves significant improvements (up to 30.6%) over its best competitor for the multi-label prediction task.

The rest of the paper is organized as follows. Section 2 presents a systematic analysis of existing network embedding methods. Section 3 describes the proposed approaches. Section 4 presents the experimental results. Section 5 reviews related work, and Section 6 concludes the paper.

2 A SYSTEMATIC ANALYSIS

In this section, we present a systematic analysis for network embedding, based on which we introduce a new, ranking-oriented design principle. This paper primarily focuses on random walk based embedding on plain networks[1]. In a nutshell, we organize existing methods, including both the existing work and our new proposed algorithms that will be shown soon, into two orthogonal dimensions: *context construction* and *objective design*. Table 1 provides a summary, where the proposed methods are in bold letters.

For the first dimension (i.e., rows of Table 1), the key idea is to construct the context network where each node can be characterized by a set of context nodes. In literature, existing methods mainly use the following three categories of context networks.

- *N1: original network*. The first category defines the connected nodes as context nodes, and directly uses the original network as the context network. Examples include SDNE [54] which applies autoencoders to reconstruct the original network, and AANE [23] which uses a graph regularization term on the original network to preserve node proximities.
- *N2: local neighborhood*. In addition to the immediate neighbors, methods in this category incorporate intermediate nearest neighbors as context nodes. For example, M-NMF [57] finds the local neighbors based on the similarities of immediate neighbors. Earlier network embedding methods such as Isomap [50] and LLE [45] also belong to this category.
- *N3: walking network*. The third category of context network is the walking network, which can be constructed by applying random walks on the original network. Typical examples in this class include DeepWalk [40], node2vec [19], GraRep [3], and SVD## [29] (which has been proved to be equivalent to random walks).

For the second dimension (i.e., columns of Table 1), the basic idea is to learn the node representations by designing an objective function to preserve certain properties of the constructed context matrix. Based on our literature review, we find that the vast majority of the existing methods follow a *pointwise* design principle, whose objective function can be decomposed into the summation of the certain goodness function over each individual edge on the context network. We further divide existing pointwise objective functions into *reconstruction-oriented* and *discrimination-oriented*.[2] More importantly, we propose to introduce *ranking-oriented* design objective into network embedding.

- *$\mathcal{J}$1: reconstruction-oriented objective function*. As the name suggests, this objective function aims to reconstruct the context network. In literature, different types of low-rank matrix approximations have been applied. For example, SVD## [29] and NetMF [41] directly use SVD, TADW [61] and HOPE [38] apply a variant of SVD by keeping the Frobenius norm, M-NMF [57] further adds non-negativity constraints, etc. In addition to low-rank matrix approximations, KL divergence [46] and autoencoders [4, 54, 64] (which first map the context matrix into embeddings and then use the embeddings to reconstruct the context matrix) have also been used.
- *$\mathcal{J}$2: discrimination-oriented objective function*. The other pointwise objective function is discrimination-oriented, whose basic idea is to distinguish the context nodes from the non-context nodes. Many methods in this category adopt the skip-gram model [37] or its variants, and typical examples include DeepWalk [40] which adopts hierarchical softmax,

[1] The generalizations of the proposed methods to (1) other types of embedding methods (e.g., graph convolution networks) and (2) other types of networks (e.g., attributed networks, heterogeneous information networks, knowledge graphs) are outside the scope of this paper, and we leave them as the further directions.

[2] Our categorization is based on how the method embeds the context network. For example, although PRUNE [28] and HOPE [38] consider node rankings in the model, they are categorized in the pointwise class as their embedding part uses discrimination-oriented and reconstruction-oriented objective functions, respectively.

Table 1: A unified framework of network embedding methods. The proposed methods are in bold letters.

Context construction \ Objective design	J1: reconstruction -oriented	J2: discrimination -oriented	J3: ranking -oriented	Combinations of J1/J2/J3
N1: original network	SVD, SDNE [54], AANE [23], LANE [24], DANE [30], SNEA [55], MVC-DNE [62], HEER [46]	LINE [49], HNE [7], PTE [48], CENE [47], EOE [59], MVE [42], IIRL [60], GATNE [6]	BPR [43], EP [14], CNE [25], **LiWine**	**PaWine**
N2: local neighborhood	SVD#, LLE [45], Isomap [50], M-NMF [57], SepNE [32]	LINE#, PRUNE [28]	BPR#, **LiWine#**	**PaWine#**
N3: walking network	SVD## [29], GraRep [3], TADW [61], HOPE [38], DNGR [4], NetMF [41], URGE [22], AROPE [66], STNE [33], NetRA [64], UltimateWalk [9]	DeepWalk [40], Planetoid [63], GENE [8], TriDNR [39], APP [67], metapath2vec [13], node2vec [19], HIN2Vec [16], struc2vec [44], SNS [34], VERSE [51], MINES [35], ANE [11], GraphSAGE [20], DP-Walker [15]	BPR##, **LiWine##**	**PaWine##**

as well as LINE [49] and node2vec [19] which adopt negative sampling to distinguish these two types of nodes.

- *J3: ranking-oriented objective function.* Different from the above pointwise objective functions which optimize over each individual edge, ranking-oriented objective functions aim to optimize over the relative ranking of a set of edges. Depending on the number of edges considered, we further divide ranking-oriented objective functions into *paiwise* and *listwise* ones.

We draw the following observations from Table 1. First of all, we can see that all the three types of context networks (rows of Table 1) have already been widely used by the existing methods; on the other hand, the existing methods have almost exclusively focused on the pointwise design (i.e., the $J1$ and $J2$ columns of Table 1). Two exceptions are EP [14] whose focus is to encode the multi-modal node attributes with a basic pairwise loss function, and CNE [25] which adds constraints on the pairwise distances between seen and unseen edges under the Bayesian framework.

More importantly, the proposed ranking-oriented design (the last two columns of Table 1) would enable a whole family of new network embedding methods. For example, the proposed LiWine in the $J3$ column encodes a listwise objective function, and PaWine augments the existing discrimination-oriented pointwise objective function with a pairwise objective function (i.e., $J2$+$J3$). Furthermore, some existing methods, even though they were originally designed for other network mining tasks, can be naturally used for network embedding. For example, BPR [43] in column $J3$, which was originally proposed for recommender systems, can be adapted as a pairwise network embedding method.

Finally, most of the methods are applicable to different types of context networks (e.g., SVD, SVD#, and SVD## are variants of SVD by applying it to the three types of context networks, respectively). For other methods (e.g., LINE[3], BPR, and the two proposed methods), we can also apply them to all the three types of context networks. We will systematically evaluate the effectiveness of these methods in the experimental section.

[3]We use the second order LINE, which can be seen as applying DeepWalk on the original network instead of the walking network [41].

3 THE PROPOSED APPROACHES

In this section, we state the problem definition and present the proposed network embedding approaches.

3.1 Problem Statement

We use $G = (V, E)$ to denote the original input network. Following conventions, we use bold capital letters for matrices. For example, $\mathbf{W}$ is used to denote the (weighted) adjacency matrix of the context network constructed from the original network. Note that $\mathbf{W}$ can be the original network. We assume that each node has two roles (i.e., central role and context role), and use two $n \times d$ matrices $\mathbf{F}$ and $\mathbf{H}$ to denote the corresponding node embedding results, respectively. We denote the i-th row of matrix $\mathbf{F}$ as $\mathbf{F}(i,:)$ which stands for the embedding of node i. With the above notations, we define the pairwise and listwise network embedding problems as follows.

Problem 1: Pairwise Network Embedding Problem. Given (1) the (weighted) adjacency matrix $\mathbf{W}$ of the context network constructed from the input network G, and (2) the set D of edge pairs where each pair contains a positive edge and a negative edge from $\mathbf{W}$; the goal is to find the node embedding matrix $\mathbf{F}$.

Problem 2: Listwise Network Embedding Problem. Given (1) the (weighted) adjacency matrix $\mathbf{W}$ of the context network constructed from the input network G, and (2) a list L_i of edges for each node i from $\mathbf{W}$; the goal is to find the node embedding matrix $\mathbf{F}$.

Different from a typical network embedding setting which is often defined with respect to the given input network, we have intentionally defined both Problem 1 and Problem 2 with respect to the *context network*. This is because our proposed algorithms (PaWine and LiWine) are orthogonal to the way the context network is constructed. In other words, they are applicable to *any* method to construct the context network (i.e., the three rows of Table 1). Throughout this paper, we refer to edges as those in the context network (as opposed to the original input network), unless stated otherwise. In the above definitions, the constructed context network could be either unweighted or weighted. The output is the embedding matrix $\mathbf{F}$. To make the proposed method compatible to some existing competitors, we could also output an additional embedding matrix $\mathbf{H}$ whose embeddings indicate the node roles as context nodes. The input of pairwise network embedding contains the set D of edge pairs in the form of (i, j) and (i, k), where i

is the central node, and $(i, j)/(i, k)$ is one of its positive/negative (e.g., observed/unobserved) edges from $\mathbf{W}$. The input of listwise network embedding contains an edge list L_i (which may contain both positive and negative edges) for each node i. Essentially, pairwise methods aim to capture the relative weights of the edge pairs, whereas listwise methods capture the relative weights of the whole edge lists.

Preliminaries: Pointwise Objective Function. Here, we discuss the existing pointwise methods, whose key idea is to design the following objective function on the context network,

$$J_{pointwise} = \sum_{(i,j)} f(\mathbf{W}(i,j), \mathbf{F}(i,;), \mathbf{F}(j,:)), \tag{1}$$

where $\mathbf{F}$ contains the embeddings we aim to learn, and the objective is a summation of separate goodness function f over each individual edge (i, j). Take DeepWalk [40] or node2vec [19] as an example. By instantiating the f function, we have the following formulation,

$$\max_{\mathbf{F}} \sum_{(i,j)} \mathbf{W}(i,j) \log \sigma(\mathbf{F}(i,:) \cdot \mathbf{F}(j,:)), \tag{2}$$

where $\cdot$ indicates inner product, $\sigma(x) = \frac{1}{1+e^{-x}}$, and we have omitted the negative sampling term for clarity. As we can see, the above equation can be decomposed into subproblems considering each edge (i, j) separately.

3.2 The Proposed PaWine

Pairwise Objective Function. In this subsection, we describe the proposed PaWine for Problem 1. We start with a basic pairwise formulation, and then augments it with a pointwise objective function.

As mentioned above, pairwise methods optimize over edge pairs of the context network. For example, given a positive edge (i, j) and a negative edge (i, k), pairwise methods aim to learn the embeddings so that the proximity between node i and j is larger than that between node i and k, i.e.,

$$J_{pairwise} = \sum_{(i,j,k)\in D} g(\mathbf{W}(i,j), \mathbf{F}(i,;), \mathbf{F}(j,:), \mathbf{F}(k,:)), \tag{3}$$

where D is the set of (i, j, k) tuples containing edge pairs ((i, j) v.s. (i, k)), and function g captures the relative weights of (i, j) and (i, k). For example, we can instantiate the following basic pairwise formulation,

$$\max_{\mathbf{F},\mathbf{H}} \sum_{(i,j,k)\in D} \mathbf{W}(i,j) \log \sigma(\mathbf{F}(i,:) \cdot \mathbf{H}(j,:) - \mathbf{F}(i,:) \cdot \mathbf{H}(k,:)), \tag{4}$$

where $\cdot$ and $\sigma(x)$ are defined in Eq. (2), and we optimize the proximity difference between edge (i, j) and edge (i, k). Compared with Eq. (3), we consider two roles of each node, i.e., central role (matrix $\mathbf{F}$) and context role (matrix $\mathbf{H}$). Here, node i plays the central role and nodes j and k play the context roles. Notice that in Eq. (2), the edge pairs are from the context network. In contrast, when applying on the original input network, the above formulation resembles the so-called Bayesian personalized ranking method [43] which was originally proposed for recommender systems. We refer to this basic method as *BPR* in this paper.

We further integrate the above pairwise formulation with the pointwise method in Eq. (2). That is, given the (i, j, k) tuple, by treating edge (i, k) as a negative sample, we have the proposed PaWine model as follows,

$$\begin{aligned}\max_{\mathbf{F},\mathbf{H}} \sum_{(i,j,k)\in D} \quad & \mathbf{W}(i,j) \log \sigma(\mathbf{F}(i,:) \cdot \mathbf{H}(j,:) - \mathbf{F}(i,:) \cdot \mathbf{H}(k,:)) \\ & + \lambda_d \mathbf{W}(i,j) [\log \sigma(\mathbf{F}(i,:) \cdot \mathbf{F}(j,:)) \\ & + \log \sigma(-\mathbf{F}(i,:) \cdot \mathbf{F}(k,:))] - \lambda(||\mathbf{F}||_F^2 + ||\mathbf{H}||_F^2),\end{aligned} \tag{5}$$

where we add a pointwise term for edge (i, j) and a negative sampling term for edge (i, k). The relative importance of these two terms are decided by parameter λ_d. The L_2 regularization term whose relative importance is controlled by parameter λ is used to keep the solution in a controllable space. Here, the regularization term is necessary to compensate for the negative sampling terms, and we need to balance them when applying the method. For example, a relatively large λ and a relatively small λ_d would lead the learned embedding values towards zeros. We will experimentally evaluate this in the next section. To instantiate the g function in Eq. (3), we choose the logistic-like function $\log \sigma(x - y)$. In addition to this loss function, other loss functions such as hinge loss, Huber loss, and WMW loss [1] can also be used. The above formulation considers two roles of each node ($\mathbf{F}$ v.s. $\mathbf{H}$). Alternatively, we could only consider the central role by substituting the $\mathbf{H}$ matrix with $\mathbf{F}$ matrix in Eq. (5). We have experimentally found that these two choices make little differences, and thus keep the $\mathbf{H}$ matrix for better understanding.

Sampling Strategy. The size of D is quadratic w.r.t to the node number n. Therefore, in practice, we usually adopt sampling strategies to sample a subset of edge pairs for each node. In this work, We consider several choices including *uniform sampling* which samples the positive edge and negative edge uniformly random, and *degree-based sampling* which samples the edge with a probability proportional to the node degree [37].

Learning Algorithm. Next, we present the learning strategy for Eq. (5). Here, since the optimization problem is non-convex, we adopt the alternating strategy. That is, we iteratively update $\mathbf{F}(i,:)$, $\mathbf{F}(j,:)$, $\mathbf{F}(k,:)$, $\mathbf{H}(j,:)$, and $\mathbf{H}(k,:)$. For example, by computing the partial derivatives, the updating rule for $\mathbf{F}(i,:)$ is as follows,

$$\begin{aligned}\mathbf{F}(i,:) \quad \leftarrow \quad & \mathbf{F}(i,:) + \eta\{\mathbf{W}(i,j)(1-\sigma_{i,j,k})(\mathbf{H}(j,:) - \mathbf{H}(k,:)) \\ & + \lambda_d \mathbf{W}(i,j)[(1-\sigma_{i,j})\mathbf{F}(j,:) \\ & - (1-\sigma_{i,k})\mathbf{F}(k,:)] - \lambda \mathbf{F}(i,:)\},\end{aligned} \tag{6}$$

where η is the learning step size, and $\sigma_{i,j,k}$, $\sigma_{i,j}$, and $\sigma_{i,k}$ are short for $\sigma(\mathbf{F}(i,:) \cdot \mathbf{H}(j,:) - \mathbf{F}(i,:) \cdot \mathbf{H}(k,:))$, $\sigma(\mathbf{F}(i,:) \cdot \mathbf{F}(j,:))$, and $\sigma(-\mathbf{F}(i,:) \cdot \mathbf{F}(k,:))$, respectively.

The PaWine algorithm is summarized in Alg. 1. In the algorithm, we iterate over each node i (Line 3), and then sample a tuple (i, j, k) for it (Line 4). Based on this tuple, we iteratively update $\mathbf{F}(i,:)$, $\mathbf{F}(j,:)$, $\mathbf{F}(k,:)$, $\mathbf{H}(j,:)$, and $\mathbf{H}(k,:)$ in each iteration (Lines 5-9). We stop the iteration when the results converge (i.e., either the learned matrices converge or a maximum iteration number is reached). Such a learning algorithm is naturally parallelizable by sampling and optimizing a mini-batch of training tuples at the same time.

Algorithm Analysis. Finally, we briefly analyze the computational complexity of the Alg. 1. The time complexity is summarized in the following lemma, which says that PaWine enjoys a linear scalability w.r.t. the data size (i.e., the node number).

Algorithm 1 The PaWine Algorithm.

Input: The adjacency matrix $\mathbf{W}$ of the context network
Output: The node embeddings $\mathbf{F}$

1: initialize $\mathbf{F}$ and $\mathbf{H}$;
2: **while** not convergent **do**
3: **for** $i = 1 : n$ **do**
4: sample a positive edge (i, j) and a negative edge (i, k);
5: update $\mathbf{F}(i,:)$ as Eq. (6);
6: $\mathbf{F}(j,:) \leftarrow \mathbf{F}(j,:) + \eta[\lambda_d \mathbf{W}(i,j)(1-\sigma_{i,j})\mathbf{F}(i,:) - \lambda \mathbf{F}(j,:)]$;
7: $\mathbf{F}(k,:) \leftarrow \mathbf{F}(k,:) + \eta[\lambda_d \mathbf{W}(i,j)(\sigma_{i,k}-1)\mathbf{F}(i,:) - \lambda \mathbf{F}(k,:)]$;
8: $\mathbf{H}(j,:) \leftarrow \mathbf{H}(j,:) + \eta[\mathbf{W}(i,j)(1-\sigma_{i,j,k})\mathbf{F}(i,:) - \lambda \mathbf{H}(j,:)]$;
9: $\mathbf{H}(k,:) \leftarrow \mathbf{H}(k,:) + \eta[\mathbf{W}(i,j)(\sigma_{i,j,k}-1)\mathbf{F}(i,:) - \lambda \mathbf{H}(k,:)]$;
10: **end for**
11: **end while**
12: **return** $\mathbf{F}$;

Lemma 1. **Time Complexity of PaWine**. *The time complexity of* PaWine *is $O(ndl)$, where n is the node number, d is the embedding dimensionality, and l is the maximum iteration number.*

Proof. The proof is omitted for brevity.

3.3 The Proposed LiWine

Listwise Objective Function. In this subsection, we describe the proposed listwise method LiWine for Problem 2. Compared to pairwise methods which consider the relative weights of edge pairs, listwise methods further consider the relative weights of edge lists, i.e., the relative rankings of a set of edges. Typically, a listwise objective function can be written as

$$J_{listwise} = \sum_{i=1}^{n} h(\sum_{(i,k)\in L_i} \mathbf{W}(i,k), \mathbf{F}(i,;), \mathbf{F}(k,:)), \tag{7}$$

where L_i is the edge list for node i, and function h captures the relative weights of this edge list. Notice that this edge list may contain multiple positive edges and negative edges.

For the listwise objective function h, we build it upon the following top-one probability function [5] which indicates the (real) probability of a given edge (i, j) being ranked in the first position of list L_i.

$$P(\mathbf{W}(i,j)) = \frac{\mathbf{W}(i,j)}{\sum_{(i,k)\in L_i} \mathbf{W}(i,k)}. \tag{8}$$

Similarly, we have the following estimated probability of a given edge being ranked in the first position,

$$\begin{aligned} Q(\hat{\mathbf{W}}(i,j)) &= Q(\mathbf{F}(i,:)\cdot\mathbf{H}(j,:)) \\ &= \frac{\sigma(\mathbf{F}(i,:)\cdot\mathbf{H}(j,:))}{\sum_{(i,k)\in L_i}\sigma(\mathbf{F}(i,:)\cdot\mathbf{H}(k,:))}, \end{aligned} \tag{9}$$

where we have generalized the original top-one probability by introducing the σ function, which might render extra flexibility of the learned embeddings. Here, we can also substitute the σ function with the *exp* function, making function Q the softmax function. In practice, we found that these two choices have similar performance in many cases, and computing σ can be significantly accelerated [37]. Therefore, we use σ function for simplicity in this paper.

Based on functions P and Q, we can have a real probability distribution and an estimated probability distribution for each edge in the edge list L_i. Then, by instantiating the h function as the cross entropy between these two probability distributions, we have our LiWine formulation,

$$\min_{\mathbf{F},\mathbf{H}} -\sum_{i=1}^{n}\sum_{(i,j)\in L_i} P(\mathbf{W}(i,j))\log[Q(\mathbf{F}(i,:)\cdot\mathbf{H}(j,:))], \tag{10}$$

where we ignore the regularization term, and functions P and Q are defined in Eq. (8) and Eq. (9), respectively. As we can see from Eq. (5) and Eq. (10), despite using different objective functions, they both follow ranking-oriented design to optimize over a set of edges. Compared with PaWine which contains parameter λ_d to balance the effect of two components, one advantage of LiWine is that it does not need such parameters.

Sampling Strategy. Similar to PaWine, we need to determine the sampling strategy to construct the list L_i for each node. There are many potential choices of sampling strategies, and we consider three of them in this work. The first one is *full sampling*, i.e., using all the possible edges (including both positive and negative ones). While this strategy has a full consideration of the edges, it would cost quadratic time making it only suitable for medium-size networks. The second sampling strategy is *uniform partial sampling*, where we keep all the positive edges and randomly sample r negative edges for each positive edge. Finally, we have the third choice *degree-based partial sampling* where we sample r negative edges for each positive edge with a probability proportional to the node degree.

Learning Algorithm. Next, we briefly present the learning algorithm for Eq. (10). To simplify the descriptions, we first define three $n \times n$ matrices whose entries are as follows,

$$\begin{aligned} \mathbf{A}(i,j) &= P(\mathbf{W}(i,j)), \\ \mathbf{B}(i,j) &= 1-\sigma(\mathbf{F}(i,:)\cdot\mathbf{H}(j,:)), \\ \mathbf{C}(i,j) &= Q(\mathbf{F}(i,:)\cdot\mathbf{H}(j,:))\mathbf{B}(i,j). \end{aligned} \tag{11}$$

In order to solve Eq. (10), we have different strategies depending on the data size. We can directly compute the partial derivative on $\mathbf{F}(i,:)$ as

$$\frac{\partial J}{\partial \mathbf{F}(i,:)} = -\sum_{(i,j)\in L_i}\mathbf{A}(i,j)[\mathbf{B}(i,j)\mathbf{H}(j,:) - \sum_{(i,k)\in L_i}\mathbf{C}(i,k)\mathbf{H}(k,:)]. \tag{12}$$

When the data size is not large, the above derivative can be further simplified. For example, if we choose the full sampling strategy, we have

$$\frac{\partial J}{\partial \mathbf{F}} = \mathbf{C}\mathbf{H} - (\mathbf{A}\odot\mathbf{B})\mathbf{H}, \tag{13}$$

where $\odot$ means the Hadamard product.

Similar to $\mathbf{F}$, we have the following partial derivative for $\mathbf{H}(j,:)$,

$$\begin{aligned} \frac{\partial J}{\partial \mathbf{H}(j,:)} &= -\sum_{i=1}^{n}\mathbf{A}(i,j)[\mathbf{B}(i,j)\mathbf{F}(i,:) - \mathbf{C}(i,j)\mathbf{F}(i,:)] \\ &= [\mathbf{A}(:,j)\odot(\mathbf{C}(:,j)-\mathbf{B}(:,j))]^T\mathbf{F}, \end{aligned} \tag{14}$$

and the following equation when we use the full sampling strategy,

$$\frac{\partial J}{\partial \mathbf{H}} = [\mathbf{A}\odot(\mathbf{C}-\mathbf{B})]^T\mathbf{F}. \tag{15}$$

Based on the above equations, the proposed LiWine algorithm is summarized in Alg. 2. As we can see, we first fix $\mathbf{H}$ and update $\mathbf{F}$ (Lines 3-6). Then, with a given sampling strategy, we sample an edge list L_j for the context node j and update $\mathbf{H}(j,:)$ accordingly with $\mathbf{F}$ fixed (Lines 7-10). The iterations are naturally parallelizable. In practice, we adopt the full sampling strategy and use Eq. (13) and Eq. (15) to substitute Lines 3-6 and Lines 7-10 when the network size

Algorithm 2 The LiWine Algorithm.

Input: The adjacency matrix $\mathbf{W}$ of the context network
Output: The node embeddings $\mathbf{F}$

```
initialize F and H;
while not convergent do
    for i = 1 : n do
        sample an edge list L_i;
        update F(i, :) as Eq. (12);
    end for
    for j = 1 : n do
        sample an edge list L_j;
        update H(j, :) as Eq. (14);
    end for
end while
return F;
```

is moderate. The algorithm ends when either the learned matrices converge or a maximum iteration number is reached.

Algorithm Analysis. The time complexity of LiWine is summarized in the following lemma, which says that LiWine enjoys a linear scalability w.r.t. the edge number of the context network.

Lemma 2. **Time Complexity of LiWine**. *The time complexity of* LiWine *is $O(mdrl)$, where m is the edge number in the context network, d is the embedding dimensionality, r is the negative sampling ratio, and l is the maximum iteration number.*

Proof. The proof is omitted for brevity.

Remarks. Note that the linear complexity in both Lemma 1 and Lemma 2 is stated in terms of the size of the context matrix (e.g., by short random walks), which might be denser than the original input network. However, as we will show in the experiment section, for both PaWine and LiWine, we often only need to use the original network as the context matrix to achieve a superior performance over the existing methods. Furthermore, even if one prefers a short random walk as context, we still do not need to explicitly construct the (potentially dense) context matrix. Instead, we can use a similar sampling strategy as DeepWalk and node2vec to obtain the pairwise or listwise context, making the proposed algorithms scalable to large networks.

Comparison with Skip-gram based Models. For skip-gram based models such as DeepWalk and node2vec as well as their origin word2vec, the basic idea is to maximize the following term,

$$\sum_{v \in V} \log Pr(N(v)|v), \tag{16}$$

where $N(v) \subset V$ is the context of the center node v. The above formulation implicitly learns the joint probability of a subset of edges of the context matrix, which resembles the high-level idea of the proposed method in this paper. However, Eq. (16) is computationally expensive or even intractable. Therefore, previous models (e.g., DeepWalk, node2vec, and word2vec) simplify Eq. (16) by imposing the conditional independence assumption, i.e.,

$$\log Pr(N(v)|v) = \sum_{u \in N(v)} \log p(u|v). \tag{17}$$

This simplification results in the pointwise objective function of these models, as their goal is to maximize the summation of each $\log(p(u|v))$ term. From this perspective, we can view the proposed ranking-oriented method as an alternative way to approximate Eq. (16). That is, our method is able to encode the interactions between different edges of the context network by a pairwise or listwise objective function. This opens the door to a whole family of network embedding methods, with enhanced empirical performance for the downstream mining tasks while still enjoying scalable computation.

Table 2: Statistics of the datasets.

Dataset	$\|V\|$	$\|E\|$	Avg. degree	# of labels
PPI	3,890	38,739	9.96	50
BlogCatalog	10,312	333,983	32.39	39
Flickr	80,513	5,899,882	73.28	195
Citerseer	3,312	4,732	1.43	6
Wiki	2,405	17,981	7.48	19

4 EXPERIMENTS

In this section, we present the experimental results.

4.1 Experimental Setup

Datasets. We conduct our experiments on five real datasets: *PPI* [19], *BlogCatalog* [19], *Flickr* [40], *Citeseer* [61], and *Wiki* [61], under three prediction tasks (multi-label prediction, clustering, and multi-class prediction). The first three datasets are widely used in the multi-label prediction task and the last two are widely used in the multi-class prediction task. All the datasets are publicly available. The statistics of the datasets are shown in Table 2. *Evaluation Metrics.* We study three prediction tasks. For the multi-label prediction task, we follow DeepWalk [40] and use the one-vs-rest logistic regression to make the predictions for all the methods. We report the resulting Micro-F1 and Macro-F1 scores. For clustering, we apply *K-means* to the embeddings to identify the clusters, and report the Normalized Mutual Information (NMI) scores. For multi-class prediction, we directly report the classification accuracy. All the reported results are the average of 10 repeating experiments.

Parameters and Implementations. For fair comparisons, all the embedding dimensionality is set to 128 unless otherwise stated. For the parameters, the sampling ratio r of LiWine is set to 1 for brevity. For the two parameters λ and λ_d in PaWine, we search over a grid $\lambda, \lambda_d \in \{0.001, 0.01, 0.1\}$ via cross-validation on the training data. For the other parameters, we either follow the default configuration or set them equally.

For the compared methods, DeepWalk and node2vec are two typical network embedding models built upon the skip-gram model. BPR is the basic pairwise method. GraphSAGE generalizes the GCN method [27] to unsupervised setting. For GraphSAGE, we need to initialize the node features. In our experiments, we initialize them with low-dimensional results from DeepWalk. VERSE is a recent proposed method that incorporates multiple relationships/similarities between nodes. AROPE is a recent method using reconstruction-oriented loss function on the walking network with arbitrary orders. For the implementations of DeepWalk, node2vec, GraphSAGE, VERSE, and AROPE, we directly use the code provided by the authors or publicly available in open libraries. For BPR, we implement it ourselves.[4]

[4]The code of the proposed algorithms is publicly available at https://github.com/SoftWiser-group/RankNE.

Table 3: Systematic evaluations of network embedding methods on the multi-label prediction task.

PPI data (Method/Micro-F1/Macro-F1)					
	J1	J2	J3		J2+J3
N1	SVD	LINE	BPR	LiWine	PaWine
	0.182	0.168	0.212	**0.237**	**0.237**
	0.127	0.146	0.182	**0.201**	**0.196**
N2	SVD#	LINE#	BPR#	LiWine#	PaWine#
	0.214	0.172	0.214	**0.234**	**0.243**
	0.178	0.153	0.183	**0.200**	**0.201**
N3	SVD##	DeepWalk	BPR##	LiWine##	PaWine##
	0.220	0.195	0.184	0.210	0.213
	0.174	0.171	0.158	0.184	0.181
BlogCatalog data (Method/Micro-F1/Macro-F1)					
	J1	J2	J3		J2+J3
N1	SVD	LINE	BPR	LiWine	PaWine
	0.301	0.352	0.405	0.413	**0.427**
	0.114	0.213	0.265	0.275	**0.282**
N2	SVD#	LINE#	BPR#	LiWine#	PaWine#
	0.299	0.364	0.393	0.409	0.404
	0.121	0.225	0.251	0.261	0.236
N3	SVD##	DeepWalk	BPR##	LiWine##	PaWine##
	0.349	0.402	0.311	0.335	0.326
	0.190	0.269	0.175	0.206	0.186
Flickr data (Method/Micro-F1/Macro-F1)					
	J1	J2	J3		J2+J3
N1	SVD	LINE	BPR	LiWine	PaWine
	0.281	0.388	0.396	**0.405**	**0.410**
	0.108	0.246	0.266	**0.281**	**0.274**
N2	SVD#	LINE#	BPR#	LiWine#	PaWine#
	0.284	0.373	0.385	0.385	0.383
	0.112	0.263	0.262	0.242	0.250
N3	SVD##	DeepWalk	BPR##	LiWine##	PaWine##
	0.365	0.397	0.365	0.372	0.351
	0.220	0.261	0.221	0.223	0.210

4.2 Systematic Evaluation Results

We first check the performance of network embedding methods according to the two dimensions in Section 2. Specially, we take the multi-label prediction task as an example, and choose one or more representative methods in each cell of Table 1. The chosen methods and their results on the first three datasets (PPI, BlogCatalog, and Flickr) are shown in Table 3. Each cell contains the method name and its Micro-F1 and Macro-F1 scores. Note that we choose the methods that can be applied on all the three context networks for comparison purpose. For example, BPR, BPR#, and BPR## stand for applying the model on the three types of context networks, respectively. For the reported results, we randomly select 50% data as training set and use the rest as test set. The reported results of PaWine are based on the parameter setting $\lambda_d = 0.1, \lambda = 0.01$ for BlogCatalog data, and $\lambda_d = 0.01, \lambda = 0.01$ for PPI and Flickr.

We can first observe from Table 3 that the best results are achieved by the proposed ranking-oriented methods. Moreover, we find that the basic pairwise method BPR has already achieved a comparable or even better result than the pointwise methods. This result indicates the high potential of ranking-oriented objective design for network embedding. Second, we can see that the best results of the pointwise methods (columns $J1$ and $J2$) are obtained on the walking network (row $N3$), whereas the best results of ranking-oriented methods (columns $J3$ and $J1 + J2$) are typically achieved on the original network (row $N1$). In other words, we usually need to construct a complex context network to ensure the performance of pointwise methods; in contrast, we can directly use the original network as context network for the ranking-oriented methods.

4.3 Effectiveness Comparison Results

Next, we compare the proposed methods with several existing methods including DeepWalk [40], node2vec [19], BPR [43], GraphSAGE [20], VERSE [51], and AROPE [66]. We choose these competitors as they address the same problem setting as our methods, i.e., embedding general networks with only the original network topology as input. The results are shown in Fig. 1.

(A) Multi-label Prediction Comparison. The Micro-F1 results of multi-label prediction are shown in Fig. 1(a) and Fig. 1(b). Similar results are observed on the Macro-F1 metrics and thus are omitted for brevity. Here, we only report the results on the PPI data and BlogCatalog data, as some of the competitors are computationally prohibitive on the Flickr data (e.g., cannot return results in 24 hours). We can observe from the figures that the proposed methods generally outperform the compared methods with different size of training data on both datasets. For example, compared with the best competitor at each training data size on PPI data, PaWine improves them by 7.4% - 30.6%, and LiWine improves them by 7.3% - 52.9%. On BlogCatalog, PaWine and LiWine improve the best competitors by up to 6.7% and 2.0%.

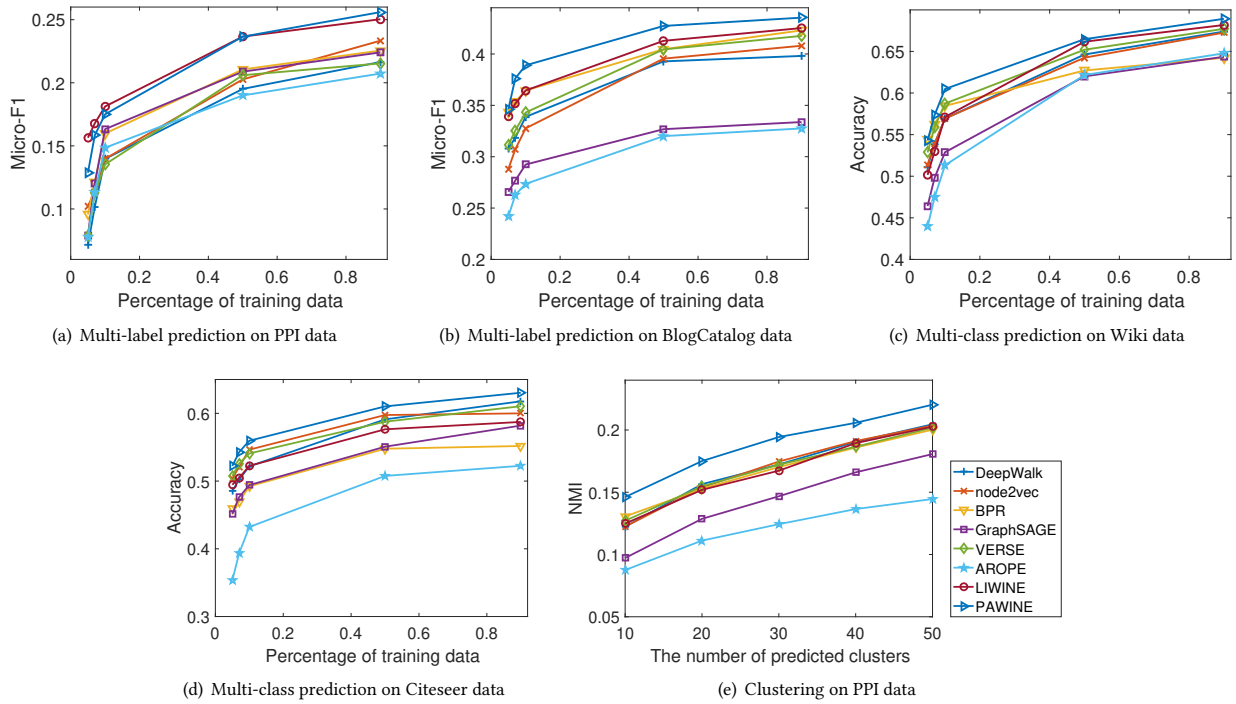
(B) Multi-class Prediction Comparison. Next, we show the effectiveness comparisons for the multi-class prediction task on the Citeseer and Wiki data. Fig. 1(c) and Fig. 1(d) show the results, where the reported results of PaWine are based on the parameter setting $\lambda_d = 0.1, \lambda = 0.1$ for Citeseer, and $\lambda_d = 0.01, \lambda = 0.01$ for Wiki. As we can see from the figures, PaWine in general has better prediction accuracy on the Citeseer data, and both PaWine and LiWine outperform the compared methods on the Wiki data. For example, on the Citeseer data, PaWine achieves 2.1% - 4.2% improvements over its competing results; on the Wiki data, PaWine improves its best competitor by up to 17.5%.

(C) Clustering Comparison. Here, we evaluate the performance of the learned embeddings in the clustering task. We set the initial cluster number from 10 to 50, and show the NMI results of PPI data in Fig. 1(e). As we can see, the proposed PaWine significantly outperforms the other methods. For example, compared with the best competitor, PaWine improves it by 7.7% - 11.9%.

Overall, the above results indicate the usefulness of ranking-oriented modeling as the baselines all adopt pointwise objective functions. Moreover, the proposed PaWine achieves significant improvements in most cases for all the three prediction tasks.

4.4 Effectiveness of LiWine as Post-Processing

Next, we evaluate the effectiveness of the proposed LiWine as a fine-tuned post-processing step for the embedding results of existing



(a) Multi-label prediction on PPI data (b) Multi-label prediction on BlogCatalog data (c) Multi-class prediction on Wiki data

(d) Multi-class prediction on Citeseer data (e) Clustering on PPI data

Figure 1: The effectiveness comparison results. The proposed methods especially PaWine outperform the compared methods in general.

Table 4: Fine-tuning results of LiWine. Each result is in the form of x/y where x means the result of the original method and y means that after applying LiWine. (• indicates the y result is significantly better than x with p-value< 0.01 and ◦ indicates no significant difference.)

	DeepWalk	node2vec	BPR	GraphSAGE	VERSE	AROPE
PPI	0.195/0.235 •	0.203/0.236 •	0.212/0.232•	0.209/0.229•	0.206/0.237•	0.190/0.227•
BlogCatalog	0.402/0.415 •	0.395/0.417 •	0.405/0.420•	0.327/0.389•	0.404/0.417•	0.320/0.378•
Wiki	0.647/0.673 •	0.643/0.669 •	0.626/0.639•	0.620/0.668•	0.651/0.678•	0.622/0.653•
Citeseer	0.591/0.603 •	0.598/0.599 ◦	0.548/0.593•	0.551/0.567•	0.588/0.608•	0.508/0.573•

methods. The results of multi-label and multi-class prediction are shown in Table 4. For brevity, we still report the results when 50% data is used as training set, and report the Micro-F1 results for multi-label prediction. Similar results are observed on other metrics and training data percentages. We do not observe significant improvements when applying PaWine as a post-processing step in most cases and thus omit the results for brevity.

We can observe from the table that after applying LiWine as fine-tuning step, most existing methods have achieved significant improvements. Additionally, some of the fine-tuned results are even better than PaWine. For example, applying LiWine after VERSE leads to 2.0% improvement compared to PaWine. These results suggest that the strength of the proposed PaWine and LiWine are complementary with each other. That is, the pairwise objective term in PaWine is more effective when integrated with the pointwise objective functions during the learning process; whereas LiWine is more effective when acting as a fine-tuned post-processing step of the existing pointwise embedding results.

4.5 Visualization

Finally, we present the visualization results of the learned embeddings. We use the Citeseer data as an example. Among the six labels, we choose three closely related ones which are harder for visualization, i.e., scientific publications belonging to the areas of 'Agents' (blue), 'AI' (orange), and 'ML' (green), respectively. We map the learned embeddings into a 2-D space with t-SNE [36], and the results of DeepWalk, LiWine, and PaWine are shown in Fig. 2.

As we can see, the result of DeepWalk is largely cluttered as publications of the same area are scattered in the space. The result of LiWine is much better as we can find some node clusters with the same color. For example, most of the 'AI' and 'Agents' publications are in the right and the bottom-right corners of the space, respectively. The result of PaWine is even better: in addition to identifying the node clusters of 'AI' and 'Agents', the 'ML' publications are concentrated at the bottom-right corner.

(a) DeepWalk (b) LiWine (c) PaWine

Figure 2: The visualization results of the Citeseer network. Color of a node indicates the area/venue of the publication. For the proposed methods (b and c), the generated embeddings with the same label are in a relatively concentrated area.

5 RELATED WORK

In this section, we briefly review the related work, including basic models and various extensions. Here, we mainly focus on the network embedding methods with explicit or implicit steps of context construction and objective function, which are most relevant to our proposed methods. In addition, there exist many other network embedding models based on convolutional neural networks [65] (e.g., ChebNet [12] and GCN [27]), recurrent neural networks (e.g., GGNN [31]), and attention mechanism (e.g., GAT [53]), etc. We refer readers to some recent surveys [2, 10, 21, 58] for more details.

Basic Models. Basic network embedding models take only the original network as the input and output the learned embeddings via context network construction and objective design. Classic examples include DeepWalk [40], LINE [49], and node2vec [19] which apply discrimination-oriented objective functions on the corresponding context networks. The reconstruction-oriented objective functions have also been widely used. For example, SVD## [29], GraRep [3], and NetMF [41] mainly focus on the context network construction and apply SVD on the constructed networks to obtain the embeddings; in contrast, SDNE [54] and DNGR [4]/NetRA [64] focus on the objective function by using autoencoders to reconstruct the original network and walking network, respectively. The proposed methods in this work fall into the category of basic models as we take the network structure as the sole input. Different from the above pointwise methods, we introduce ranking-oriented design, and further propose pairwise and listwise methods for network embedding.

Variants and Extensions. In addition to the basic models, many variants and extensions have been proposed. The first extension aims at collectively modeling the network structure and the *node attributes*. Examples include [7, 17, 23, 24, 30, 33, 47, 55] which collectively learn and integrate the representations of node attributes, and [20, 61–63] which develop inductive models to predict the embeddings based on the node attributes. The second extension is to incorporate the *community structure* for network embedding when such information is available [8, 57]. The third extension pays special attention to *directed networks* by, for example, computing asymmetric proximities between nodes [38, 67]. The fourth extension is specially designed for *signed networks* (with both positive links and negative links), where the key issue is to deal with the negative links [26, 55, 56]. Although some of these signed network embedding methods also use pairwise objective functions, they bear some subtle differences from the proposed PaWine algorithm, in the sense that PaWine (1) deals with the general input network with non-negative link weights and (2) applies the pairwise objective function on the *context network* (as opposed to the input network) in conjuntion with the pointwise objective function. The fifth extension considers the node embedding problem in *heterogeneous networks* which contain nodes/edges of different types [7, 13, 16, 46, 59]. For example, metapath2vec [13] and HIN2Vec [16] define the context network with metapaths over the heterogeneous networks, and apply skip-gram model and logistic classifier to learn the embeddings, respectively. The sixth extension designs *semi-supervised* network embedding models by incorporating the supervision information of specific prediction tasks [18, 24, 27, 39, 48, 52, 63]. For example, PTE [48] and TriNDR [39] incorporate the known labels when learning the embeddings, and use these embeddings to predict the unknown labels. The seventh extension targets at the *dynamics* of networks [30, 68, 69]. Although not the focus of this work, the proposed ranking-oriented methods are potentially applicable to all these extensions and variants.

6 CONCLUSIONS

In this paper, we divide existing network embedding methods into two stages of context construction and objective design, and propose the ranking-oriented design principle for network embedding. We further instantiate two new network embedding algorithms, including a pairwise network embedding method PaWine which optimizes the relative weights of edge pairs in conjunction with a pointwise objective function, and a listwise method LiWine which optimizes the relative weights of edge lists. Both proposed methods have a linear time complexity with respect to the size of the context network. Comprehensive experimental evaluations demonstrate the effectiveness of the proposed approaches. Future work includes generalizing the ranking-oriented network embedding to (1) other types of embedding methods (e.g., graph convolution networks) and (2) other types of networks (e.g., attributed networks, heterogeneous information networks, knowledge graphs).

ACKNOWLEDGEMENTS

This work is supported by the National Natural Science Foundation of China (No.61690204, 61932021, 61672274), and the Collaborative

Innovation Center of Novel Software Technology and Industrialization. Hanghang Tong is partially supported by NSF (1947135, 2003924, and 1939725). Yuan Yao is the corresponding author.

REFERENCES

[1] Lars Backstrom and Jure Leskovec. 2011. Supervised random walks: predicting and recommending links in social networks. In *WSDM*. ACM, 635–644.
[2] Hongyun Cai, Vincent W Zheng, and Kevin Chang. 2018. A comprehensive survey of graph embedding: problems, techniques and applications. *TKDE* (2018).
[3] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning graph representations with global structural information. In *CIKM*. 891–900.
[4] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2016. Deep neural networks for learning graph representations. In *AAAI*. 1145–1152.
[5] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: from pairwise approach to listwise approach. In *ICML*. ACM, 129–136.
[6] Yukuo Cen, Xu Zou, Jianwei Zhang, Hongxia Yang, Jingren Zhou, and Jie Tang. 2019. Representation Learning for Attributed Multiplex Heterogeneous Network. In *KDD9*.
[7] 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.
[8] Jifan Chen, Qi Zhang, and Xuanjing Huang. 2016. Incorporate group information to enhance network embedding. In *CIKM*. 1901–1904.
[9] Siheng Chen, Sufeng Niu, Leman Akoglu, Jelena Kovačević, and Christos Faloutsos. 2017. Fast, Warped Graph Embedding: Unifying Framework and One-Click Algorithm. *arXiv preprint arXiv:1702.05764* (2017).
[10] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. 2018. A survey on network embedding. *TKDE* (2018).
[11] Quanyu Dai, Qiang Li, Jian Tang, and Dan Wang. 2018. Adversarial Network Embedding. In *AAAI*.
[12] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*.
[13] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable Representation Learning for Heterogeneous Networks. In *KDD*.
[14] Alberto Garcia Duran and Mathias Niepert. 2017. Learning Graph Representations with Embedding Propagation. In *NIPS*. 5125–5136.
[15] Rui Feng, Yang Yang, Wenjie Hu, Fei Wu, and Yueting Zhuang. 2018. Representation Learning for Scale-free Networks. In *AAAI*.
[16] Tao-yang Fu, Wang-Chien Lee, and Zhen Lei. 2017. HIN2Vec: Explore Meta-paths in Heterogeneous Information Networks for Representation Learning. In *CIKM*. ACM, 1797–1806.
[17] Hongchang Gao and Heng Huang. 2018. Deep Attributed Network Embedding.. In *IJCAI*. 3364–3370.
[18] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. 2018. Large-Scale Learnable Graph Convolutional Networks. In *KDD*. 1416–1424.
[19] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *KDD*. 855–864.
[20] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS*. 1025–1035.
[21] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. *arXiv* (2017).
[22] Jiafeng Hu, Reynold Cheng, Zhipeng Huang, Yixang Fang, and Siqiang Luo. 2017. On embedding uncertain graphs. In *CIKM*. ACM, 157–166.
[23] Xiao Huang, Jundong Li, and Xia Hu. 2017. Accelerated attributed network embedding. In *SDM*.
[24] Xiao Huang, Jundong Li, and Xia Hu. 2017. Label Informed Attributed Network Embedding. In *WSDM*.
[25] Bo Kang, Jefrey Lijffijt, and Tijl De Bie. 2019. Conditional Network Embeddings. In *ICLR*.
[26] Junghwan Kim, Haekyu Park, Ji-Eun Lee, and U Kang. 2018. SIDE: Representation Learning in Signed Directed Networks. In *WWW*. 509–518.
[27] Thomas Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.
[28] Yi-An Lai, Chin-Chi Hsu, Wen Hao Chen, Mi-Yen Yeh, and Shou-De Lin. 2017. PRUNE: Preserving Proximity and Global Ranking for Network Embedding. In *NIPS*. 5263–5272.
[29] Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. In *NIPS*. 2177–2185.
[30] 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*.
[31] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2016. Gated graph sequence neural networks. In *ICLR*.
[32] Ziyao Li, Liang Zhang, and Guojie Song. 2019. Sepne: Bringing separability to network embedding. In *AAAI*.
[33] Jie Liu, Zhicheng He, Lai Wei, and Yalou Huang. 2018. Content to node: Self-translation network embedding. In *KDD*. 1794–1802.
[34] Tianshu Lyu, Yuan Zhang, and Yan Zhang. 2017. Enhancing the Network Embedding Quality with Structural Similarity. In *CIKM*. ACM, 147–156.
[35] Yao Ma, Zhaochun Ren, Ziheng Jiang, Jiliang Tang, and Dawei Yin. 2018. Multi-Dimensional Network Embedding with Hierarchical Structure. In *WSDM*.
[36] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research* 9, Nov (2008), 2579–2605.
[37] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv* (2013).
[38] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric Transitivity Preserving Graph Embedding. In *KDD*. 1105–1114.
[39] Shirui Pan, Jia Wu, Xingquan Zhu, Chengqi Zhang, and Yang Wang. 2016. Tri-party deep network representation. In *IJCAI*. 1895–1901.
[40] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *KDD*. 701–710.
[41] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network Embedding as Matrix Factorization: UnifyingDeepWalk, LINE, PTE, and node2vec. In *WSDM*.
[42] Meng Qu, Jian Tang, Jingbo Shang, Xiang Ren, Ming Zhang, and Jiawei Han. 2017. An Attention-based Collaboration Framework for Multi-View Network Representation Learning. In *CIKM*. ACM, 1767–1776.
[43] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *UAI*.
[44] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. 2017. struc2vec: Learning node representations from structural identity. In *KDD*.
[45] Sam T Roweis and Lawrence K Saul. 2000. Nonlinear dimensionality reduction by locally linear embedding. *Science* (2000), 2323–2326.
[46] Yu Shi, Qi Zhu, Fang Guo, Chao Zhang, and Jiawei Han. 2018. Easing Embedding Learning by Comprehensive Transcription of Heterogeneous Information Networks. In *KDD*. 2190–2199.
[47] Xiaofei Sun, Jiang Guo, Xiao Ding, and Ting Liu. 2016. A General Framework for Content-enhanced Network Representation Learning. *arXiv* (2016).
[48] Jian Tang, Meng Qu, and Qiaozhu Mei. 2015. Pte: Predictive text embedding through large-scale heterogeneous text networks. In *KDD*. 1165–1174.
[49] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW*. 1067–1077.
[50] Joshua B Tenenbaum, Vin De Silva, and John C Langford. 2000. A global geometric framework for nonlinear dimensionality reduction. *Science* (2000).
[51] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. 2018. VERSE: Versatile Graph Embeddings from Similarity Measures. In *WWW*.
[52] Cunchao Tu, Weicheng Zhang, Zhiyuan Liu, and Maosong Sun. 2016. Max-margin DeepWalk: discriminative learning of network representation. In *IJCAI*.
[53] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.
[54] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *KDD*. 1225–1234.
[55] Suhang Wang, Charu Aggarwal, Jiliang Tang, and Huan Liu. 2017. Attributed signed network embedding. In *CIKM*. ACM, 137–146.
[56] Suhang Wang, Jiliang Tang, Charu Aggarwal, Yi Chang, and Huan Liu. 2017. Signed network embedding in social media. In *SDM*.
[57] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. 2017. Community Preserving Network Embedding. In *AAAI*. 203–209.
[58] Yaojing Wang, Yuan Yao, Hanghang Tong, Feng Xu, and Jian Lu. 2019. A Brief Review of Network Embedding. *BDMA* 2, 1 (2019), 35–47.
[59] Linchuan Xu, Xiaokai Wei, Jiannong Cao, and Philip S Yu. 2017. Embedding of Embedding (EOE): Joint Embedding for Coupled Heterogeneous Networks. In *WSDM*.
[60] Linchuan Xu, Xiaokai Wei, Jiannong Cao, and Philip S Yu. 2018. On Exploring Semantic Meanings of Links for Embedding Social Networks. In *WWW*. 479–488.
[61] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y Chang. 2015. Network representation learning with rich text information. In *IJCAI*. 2111–2117.
[62] Dejian Yang, Senzhang Wang, Chaozhuo Li, Xiaoming Zhang, and Zhoujun Li. 2017. From Properties to Links: Deep Network Embedding on Incomplete Graphs. In *CIKM*. ACM.
[63] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. 2016. Revisiting Semi-Supervised Learning with Graph Embeddings. In *ICML*. 40–48.
[64] Wenchao Yu, Cheng Zheng, Wei Cheng, Charu C Aggarwal, Dongjin Song, Bo Zong, Haifeng Chen, and Wei Wang. 2018. Learning deep network representations with adversarially regularized autoencoders. In *KDD*. 2663–2671.
[65] Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. 2018. Graph Convolutional Networks: Algorithms, Applications and Open Challenges. In *CSoNet*.
[66] Ziwei Zhang, Peng Cui, Xiao Wang, Jian Pei, Xuanrong Yao, and Wenwu Zhu. 2018. Arbitrary-Order Proximity Preserved Network Embedding. In *KDD*.
[67] Chang Zhou, Yuqiong Liu, Xiaofei Liu, Zhongyi Liu, and Jun Gao. 2017. Scalable Graph Embedding for Asymmetric Proximity. In *AAAI*. 2942–2948.
[68] Lekui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. 2018. Dynamic Network Embedding by Modeling Triadic Closure Process. In *AAAI*.
[69] Yuan Zuo, Guannan Liu, Hao Lin, Jia Guo, Xiaoqian Hu, and Junjie Wu. 2018. Embedding Temporal Network via Neighborhood Formation. In *KDD*.