

# Billion-scale Network Embedding with Iterative Random Projection

Ziwei Zhang, Peng Cui, Haoyang Li, Xiao Wang and Wenwu Zhu  
Department of Computer Science and Technology, Tsinghua University, China  
zw-zhang16@mails.tsinghua.edu.cn, cuip@tsinghua.edu.cn, lihaoyang96@gmail.com  
wangxiao007@mail.tsinghua.edu.cn, wwzhu@tsinghua.edu.cn

**Abstract**—Network embedding, which learns low-dimensional vector representation for nodes in the network, has attracted considerable research attention recently. However, the existing methods are incapable of handling billion-scale networks, because they are computationally expensive and, at the same time, difficult to be accelerated by distributed computing schemes. To address these problems, we propose RandNE (Iterative Random Projection Network Embedding), a novel and simple billion-scale network embedding method. Specifically, we propose a Gaussian random projection approach to map the network into a low-dimensional embedding space while preserving the high-order proximities between nodes. To reduce the time complexity, we design an iterative projection procedure to avoid the explicit calculation of the high-order proximities. Theoretical analysis shows that our method is extremely efficient, and friendly to distributed computing schemes without any communication cost in the calculation. We also design a dynamic updating procedure which can efficiently incorporate the dynamic changes of the networks without error aggregation. Extensive experimental results demonstrate the efficiency and efficacy of RandNE over state-of-the-art methods in several tasks including network reconstruction, link prediction and node classification on multiple datasets with different scales, ranging from thousands to billions of nodes and edges.

**Index Terms**—Network Embedding, High-order Proximity, Billion-Scale, Dynamic Networks, Distributed Computing

## I. INTRODUCTION

Network embedding is an emerging research topic in recent years, aiming to represent nodes by low-dimensional vectors while maintaining the structures and properties of the network [1]. Many methods have been proposed for network embedding, such as using random walks [2], matrix factorization [3] and deep learning [4]. With these methods, many network analysis tasks can be fulfilled in vector spaces and benefit from off-the-shelf machine learning models.

Despite such progress, the targeted networks of the existing methods are often in thousand or million scale. However, many real networks have billions of nodes and edges, such as social networks, e-commerce networks and the Internet. The billion-scale networks pose great computational challenges to the existing methods. The bottleneck lies in that the existing methods are all learning-based and thus involve computationally expensive optimization procedures. For example, Stochastic Gradient Descend (SGD) is a commonly used optimization method in network embedding [5], but it requires a great number of iterations to converge, which may not be feasible for billion-scale networks. One way to accelerate is to resort to

distributed computing solutions, but the optimization methods, like SGD, often require global embedding information for searching gradients, leading to intense communication cost. As a result, how to design an efficient and effective billion-scale network embedding method that is friendly to distributed computing is still an open problem.

Different from learning-based methods, random projection is a simple and powerful technique to form low-dimensional embedding spaces while preserving the structures of the original space. It is also friendly to distributed computing, and thus widely exploited in large-scale data scenarios [6]. But the extremely sparse structures of real networks pose great challenges to applying random projection to network embedding. The existing work [1] has demonstrated that high-order proximities between nodes are essential to be preserved in network embedding and can effectively address the sparsity issue. Hence, how to design a high-order proximity preserved random projection method while maintaining the efficiency of the method is the key problem of billion-scale network embedding.

In this paper, we propose RandNE<sup>1</sup> (Iterative Random Projection Network Embedding), a novel and simple billion-scale network embedding method based on high-order proximity preserved random projection. Specifically, we propose using Gaussian random projection to minimize the matrix factorization objective function of preserving the high-order proximity. In order to avoid the explicit calculation of high-order proximities which induces high computational complexities, we design an iterative projection procedure, enabling arbitrary high-order proximity preserved random projection with a linear time complexity. Theoretical analysis is provided to guarantee that i) RandNE is much more computationally efficient than the existing methods, ii) it can well support distributed computing without any communication cost between different servers in the calculation, and iii) it can efficiently incorporate the dynamic changes of the networks without error aggregation. These merits make RandNE a promising solution for billion-scale network embedding, even in dynamic environments.

Extensive experiments are conducted in network reconstruction, link prediction and node classification tasks on multiple datasets with different scales, ranging from thousands to billions of nodes and edges. The results show that RandNE can

<sup>1</sup>The code is available at <https://github.com/ZW-ZHANG/RandNE>.

boost the efficiency of network embedding by about 2 orders of magnitude over state-of-the-art methods<sup>2</sup> while achieving a superior or comparable accuracy. For the WeChat<sup>3</sup> network with 250 millions nodes and 4.8 billion edges, RandNE can produce 512-dimensional embeddings within 7 hours with 16 distributed servers.

The contributions of our paper are summarized as follows:

- We propose RandNE, a novel and simple random projection based network embedding method that enables billion-scale network embedding.
- We design an iterative projection procedure to realize high-order proximities preserved random projection efficiently without explicitly calculating the high-order proximities.
- We theoretically and empirically prove that RandNE can well support distributed computing without communication cost and can efficiently deal with dynamic networks without error aggregation.

The rest of this paper is organized as follows. In Section 2, we briefly review related works. We give our problem formulation in Section 3 and introduce our proposed method in Section 4. Experimental results are reported in Section 5. Finally, we summarize in Section 6.

## II. RELATED WORK

Network embedding has attracted considerable research attention in the past few years, aiming to bridge the gap between network analysis and off-the-shelf machine learning techniques. Here, we briefly review some representative network embedding methods, and readers are referred to [1] for a comprehensive survey.

The flourish of network embedding research begins when DeepWalk [2] first proposes using truncated random walks to explore the network structure and utilizes the skip-gram model [7] from word embedding to derive the embedding vectors of nodes. LINE [5] takes a similar idea with an explicit objective function by setting the walk length as one, and introduces the negative sampling strategy [8] to accelerate the training procedure. Node2vec [9] generalizes these two methods by taking potentially biased random walks for more flexibility. These random walks based methods are proven equivalent to factorizing a high-order proximity matrix [10].

On the other hand, explicit matrix factorization methods have been proposed for network embedding. GraRep [11] directly applies SVD to preserve high-order proximity matrices. HOPE [12] proposes using generalized SVD to preserve the asymmetric transitivity in directed networks. Community structure, an important mesoscopic structure of the network, is preserved by non-negative matrix factorization in [13]. [14] introduces a unified framework for matrix factorization and utilizes a sparsification technique to speed up SVD. Another approximate matrix factorization technique is introduced

in [15]. AROPE [3] improves these works by preserving arbitrary-order proximity simultaneously.

Deep learning model is also applied to network embedding. SDNE [4] first considers the high non-linearity in network embedding and proposes a deep auto-encoder to preserve the first two order proximities. DHNE [16] extends this framework for preserving the indecomposability in hyper-networks.

Besides static networks, how to embed dynamic networks where nodes and edges change over time also attracts research attention. DHPE [17] and DANE [18] propose using matrix perturbation to handle the changes of edges. DepthLGP adopts a Gaussian process to handle out-of-sample nodes. DynamicTriad [19] considers the triangle closure characteristic of network evolving.

Despite their remarkable performance, these methods are all learning-based and the targeted networks are often in thousand or million scale. In [20], a modification of DeepWalk is applied to a billion-scale network aliItemGraph. However, their method has the same time complexity as DeepWalk, which is much more computationally expensive than our method by over two orders of magnitude (see Figure 1 in the experiments section). Besides, it does not address the problem of distributed computing or handling dynamic networks.

How to embed networks with side information is also explored. For instance, [21], [22] utilize metapaths to embed heterogenous information networks where node and edge types are available. Node attributes and node labels are taken into consideration in [18], [23] and [24]–[26] respectively. In this paper, we focus on the most fundamental case that only the network structure is available.

Another closely related topic is random projection [6], [27]–[29], which is widely adopted in dimension reduction. But existing random projection methods do not consider the sparsity problem in network embedding, and thus cannot be directly applied.

## III. NOTATIONS AND PROBLEM FORMULATION

### A. Notations

First, we summarize the notations used in this paper. For a network  $G$  with  $N$  nodes and  $M$  edges, we use  $\mathbf{A}$  to denote the adjacency matrix. In this paper, we mainly consider undirected networks, so  $\mathbf{A}$  is symmetric.  $\mathbf{A}(i, :)$  and  $\mathbf{A}(:, j)$  denote its  $i^{\text{th}}$  row and  $j^{\text{th}}$  column respectively.  $\mathbf{A}(i, j)$  is the element in the  $i^{\text{th}}$  row and  $j^{\text{th}}$  column.  $\mathbf{A}^T$  denotes the transpose of  $\mathbf{A}$ . Throughout the paper, we use bold uppercase characters to denote matrices and bold lowercase characters to denote vectors, e.g.  $\mathbf{X}$  and  $\mathbf{x}$  respectively. We use dot to denote the matrix product of two matrices, e.g.  $\mathbf{B} \cdot \mathbf{C}$ . Functions are marked by curlicue, e.g.  $\mathcal{F}(\cdot)$ .

### B. Problem Formulation

To represent nodes in a network by low-dimensional vectors, one commonly adopted objective function in network embedding is matrix factorization, which decomposes a targeted similarity function of the adjacency matrix  $\mathcal{F}(\mathbf{A}) \in \mathbb{R}^{N \times N}$  into

<sup>2</sup>These algorithms are tested using the source code published by their authors.

<sup>3</sup>One of the largest social network platforms in China.

the product of two low-dimensional matrices  $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{N \times d}$  with the following objective function:

$$\min_{\mathbf{U}, \mathbf{V}} \|\mathcal{F}(\mathbf{A}) - \mathbf{U} \cdot \mathbf{V}^T\|_p, \quad (1)$$

where  $p$  is the norm and  $d$  is the dimensionality of the embedding. In this paper, we only consider undirected networks and symmetric similarities, so  $\mathbf{U} = \mathbf{V}$ . We also focus on the spectral norm, i.e.  $p = 2$ , which is widely adopted [30]. The adjacency matrix  $\mathbf{A}$  could be replaced by other variants, such as the Laplacian matrix or transition matrix [10]. Here, we focus on the adjacency matrix unless stated otherwise.

The previous work has shown that high-order proximities are essential to be preserved in network embedding, which can be formulated as a polynomial function of the adjacency matrix [14], [15]. In this paper, we assume that  $\mathcal{F}(\mathbf{A})$  is a positive semi-definite function, so it can be formulated as  $\mathcal{F}(\mathbf{A}) = \mathbf{S} \cdot \mathbf{S}^T$ . Then, we can rewrite Eq. (1) as:

$$\min_{\mathbf{U}} \|\mathbf{S} \cdot \mathbf{S}^T - \mathbf{U} \cdot \mathbf{U}^T\|_2$$

$$\mathbf{S} = \alpha_0 \mathbf{I} + \alpha_1 \mathbf{A} + \alpha_2 \mathbf{A}^2 + \dots + \alpha_q \mathbf{A}^q, \quad (2)$$

where  $\mathbf{S}$  is the high-order proximity matrix,  $\alpha_0, \alpha_1, \dots, \alpha_q$  are pre-defined weights and  $q$  is the order.

From Eckart-Young theorem [31], it is well known that Singular Value Decomposition (SVD) can lead to the optimal solution of Eq. (2). However, SVD is computationally expensive and thus not suitable for large-scale networks.

#### IV. RANDNE: THE PROPOSED METHOD

##### A. Gaussian Random Projection Embedding

To minimize the objective function in Eq. (2), an extremely simple yet effective method is random projection, and Gaussian random projection is widely used [6]. Formally, let  $\mathbf{R} \in \mathbb{R}^{N \times d}$  and each element of  $\mathbf{R}$  follows an i.i.d Gaussian distribution  $\mathbf{R}(i, j) \sim \mathcal{N}(0, \frac{1}{d})$ . Then, the embeddings  $\mathbf{U}$  can be obtained by performing a matrix product:

$$\mathbf{U} = \mathbf{S} \cdot \mathbf{R} = (\alpha_0 \mathbf{I} + \alpha_1 \mathbf{A} + \alpha_2 \mathbf{A}^2 + \dots + \alpha_q \mathbf{A}^q) \mathbf{R}, \quad (3)$$

i.e. we randomly project the proximity matrix  $\mathbf{S}$  into a low-dimensional subspace. Gaussian random projection has an theoretical guarantee, as we specific in the following theorem.

**Theorem 1.** *For any similarity matrix  $\mathbf{S}$ , denote its rank as  $r_S$ . Then, for any  $\epsilon \in (0, \frac{1}{2})$ , the following equation holds:*

$$P \left[ \|\mathbf{S} \cdot \mathbf{S}^T - \mathbf{U} \cdot \mathbf{U}^T\|_2 > \epsilon \|\mathbf{S}^T \cdot \mathbf{S}\|_2 \right] \leq 2r_S e^{-\frac{(\epsilon^2 - \epsilon^3)d}{4}}, \quad (4)$$

where  $\mathbf{U} = \mathbf{S} \cdot \mathbf{R}$  and  $\mathbf{R}$  is a Gaussian random matrix.

*Proof:* See appendix. ■

The theorem basically shows that the residual of our projection  $\mathbf{S} \cdot \mathbf{S}^T - \mathbf{U} \cdot \mathbf{U}^T$  has a much smaller spectral radius compared to the spectral radius of the original high-order proximity  $\mathbf{S} \cdot \mathbf{S}^T$ . In other words, the embedding  $\mathbf{U}$  captures the ‘‘core component’’ of the high-order proximity. As a result, performing a Gaussian random projection can

---

#### Algorithm 1 RandNE: Iterative Random Projection Network Embedding

---

**Require:** Adjacency Matrix  $\mathbf{A}$ , Dimensionality  $d$ , Order  $q$ ,  
Weights  $\alpha_0, \alpha_1, \dots, \alpha_q$

**Ensure:** Embedding Results  $\mathbf{U}$

- 1: Generate  $\mathbf{R} \in \mathbb{R}^{N \times d} \sim \mathcal{N}(0, \frac{1}{d})$
  - 2: Perform a Gram Schmidt process on  $\mathbf{R}$  to obtain the orthogonal projection matrix  $\mathbf{U}_0$
  - 3: **for**  $i$  in  $1:q$  **do**
  - 4:   Calculate  $\mathbf{U}_i = \mathbf{A} \cdot \mathbf{U}_{i-1}$
  - 5: **end for**
  - 6: Calculate  $\mathbf{U} = \alpha_0 \mathbf{U}_0 + \alpha_1 \mathbf{U}_1 + \dots + \alpha_q \mathbf{U}_q$
- 

effectively minimize the objective function in Eq. (2). Actually, Gaussian random projection is also known to have other merits, such as preserving the margin for classification [28], which we omit for brevity.

For the projection matrix, it is proven that orthogonal Gaussian random matrix can further improve the accuracy, which can be easily obtained by performing a Gram Schmidt process on the columns of the Gaussian random matrix [29]. In this paper, we use the orthogonal Gaussian random matrix as the projection matrix.

However, since  $\mathbf{S}$  may not be a sparse matrix, directly calculating  $\mathbf{S}$  and performing the projection is still time consuming and not scalable to large-scale networks.

##### B. Iterative Projection

To address the efficiency problem, we design an iterative projection procedure to avoid the explicit calculation of the high-order proximity matrix  $\mathbf{S}$ . Specifically, from Eq. (3), we can decompose  $\mathbf{U}$  into matrices of different orders:

$$\mathbf{U} = \alpha_0 \mathbf{U}_0 + \alpha_1 \mathbf{U}_1 + \dots + \alpha_q \mathbf{U}_q, \quad (5)$$

where  $\mathbf{U}_i = \mathbf{A}^i \cdot \mathbf{R}$ ,  $0 \leq i \leq q$ . Then, the decomposed parts,  $\mathbf{U}_1 \dots \mathbf{U}_q$ , can be calculated iteratively:

$$\mathbf{U}_i = \mathbf{A} \cdot \mathbf{U}_{i-1}, \quad \forall 1 \leq i \leq q. \quad (6)$$

Note that in Eq. (6), we only need to calculate the matrix product of the adjacency matrix and a low-dimensional matrix. Since the adjacency matrix is sparse, we can use sparse matrix products, which are highly scalable and efficient.

##### C. Time Complexity and Distributed Computing

We show our algorithm framework in Algorithm 1. Then, we analyze the time complexity of Algorithm 1. The complexity of line 1 is  $O(N \cdot d)$ , the complexity of line 2 is  $O(N \cdot d^2)$ , the complexity of each iteration from line 3 to line 5 is  $O(M \cdot d)$  and the complexity of line 6 is  $O(q \cdot N \cdot d)$ , where  $N$  and  $M$  are the number of nodes and edges in the network respectively,  $q$  is the preset order and  $d$  is the dimensionality of the embedding space. As a result, the overall time complexity is  $O(N \cdot d^2 + M \cdot q \cdot d)$ , i.e. our method is linear with respect to the network size.

---

**Algorithm 2** Distributed Calculation of RandNE

---

**Require:** Adjacency matrix  $\mathbf{A}$ , Initial Projection  $\mathbf{U}_0$ , Parameters of RandNE,  $K$  Distributed Servers

**Ensure:** Embedding Results  $\mathbf{U}$

- 1: Broadcast  $\mathbf{A}$ ,  $\mathbf{U}_0$  and parameters into  $K$  servers
  - 2: Set  $i = 1$
  - 3: **repeat**
  - 4:   **if** There is an idle server  $k$  **then**
  - 5:     Calculate  $\mathbf{U}(i, :)$  in server  $k$
  - 6:      $i = i + 1$
  - 7:   Gather  $\mathbf{U}(i, :)$  from server  $k$  after calculation
  - 8:   **end if**
  - 9: **until**  $i > d$
  - 10: Return  $\mathbf{U}$
- 

From the above analysis, we can also see that our method is extremely efficient because it only needs to iterate  $q$  times, and within each iteration, only a simple matrix product needs to be calculated. In contrast, although some existing network embedding methods are also proven to have linear time complexities, such as the embedding methods based on SGD [5] or SVD [12], they inevitably need dozens or hundreds of iterations in the optimization. As a result, our method is more efficient than these methods by orders of magnitude, and is thus more suitable for billion-scale network embedding.

In addition, according to the property of matrix products, each dimension (i.e. column) of  $\mathbf{U}_i$  can be calculated separately without any information from other dimensions. We formalize this property in the following theorem.

**Theorem 2.** *For any  $j \neq l$ , the calculation of  $\mathbf{U}_i(:, j)$  and  $\mathbf{U}_i(:, l)$ ,  $1 \leq i \leq q$  from line 3 to line 5 in Algorithm 1 are independent, if  $\mathbf{A}$  and  $\mathbf{U}_0$  are known to all the servers.*

*Proof:* Straightforward from the property of matrix products. ■

The theorem shows that our method naturally supports distributed computing by allocating the calculation of different dimensions into distributed servers, and no communication is needed during the calculation process, if  $\mathbf{A}$  and  $\mathbf{U}_0$  are known to all the servers. We design a simple distributed protocol using the theorem, as specified in Algorithm 2.

For networks that cannot be stored in the memory or when the number of servers exceeds the dimensionality, we can use more advanced distributed matrix multiplication algorithms, such as [32], [33], which we leave as the future work. This is in contrast with the existing methods which can only be parallelizable within one server but are hard to be distributed because of communication cost. This merit lays another foundation for applying RandNE to billion-scale networks.

#### D. Dynamic Updating

As many real networks are dynamic, we next show how to efficiently update RandNE to incorporate dynamic changes.

First, we focus on the changes of edges. From Algorithm 1, to update the final embedding vectors, we only need to update

---

**Algorithm 3** Dynamic Updating of RandNE

---

**Require:** Adjacency Matrix  $\mathbf{A}$ , Dynamic Changes  $\Delta\mathbf{A}$ , Previous Projection Results  $\mathbf{U}_0, \mathbf{U}_1, \dots, \mathbf{U}_q$

**Ensure:** Updated Projection Results  $\mathbf{U}'_0, \mathbf{U}'_1, \dots, \mathbf{U}'_q$

- 1: **if**  $\Delta\mathbf{A}$  includes  $N'$  new nodes **then**
  - 2:   Generate an orthogonal projection  $\hat{\mathbf{U}}_0 \in \mathbb{R}^{N' \times d}$
  - 3:   Concatenate  $\hat{\mathbf{U}}_0$  with  $\mathbf{U}_0$  to obtain  $\mathbf{U}'_0$
  - 4:   Add  $N'$  all-zero rows in  $\mathbf{U}_1 \dots \mathbf{U}_q$
  - 5: **end if**
  - 6: Set  $\Delta\mathbf{U}_0 = 0$
  - 7: **for**  $i$  in  $1:q$  **do**
  - 8:   Calculate  $\Delta\mathbf{U}_i$  using Eq. (7)
  - 9:   Calculate  $\mathbf{U}'_i = \mathbf{U}_i + \Delta\mathbf{U}_i$
  - 10: **end for**
- 

the decomposed parts  $\mathbf{U}_i, 1 \leq i \leq q$ . Formally, we denote the changes in the adjacency matrix as  $\Delta\mathbf{A}$  and the changes in  $\mathbf{U}_i$  as  $\Delta\mathbf{U}_i, 1 \leq i \leq q$ . From Eq. (6), we have:

$$\begin{aligned} \mathbf{U}_i + \Delta\mathbf{U}_i &= (\mathbf{A} + \Delta\mathbf{A}) \cdot (\mathbf{U}_{i-1} + \Delta\mathbf{U}_{i-1}) \\ \Rightarrow \Delta\mathbf{U}_i &= \mathbf{A} \cdot \Delta\mathbf{U}_{i-1} + \Delta\mathbf{A} \cdot \mathbf{U}_{i-1} + \Delta\mathbf{A} \cdot \Delta\mathbf{U}_{i-1}. \end{aligned} \quad (7)$$

Then, we can iteratively calculate  $\Delta\mathbf{U}_i$  using Eq. (7).

Besides, nodes in the network may also be added or deleted. As for deleting nodes, it can be treated equivalently as the changes of edges by deleting all edges of the deleted nodes. For newly added nodes, we first add some empty nodes (i.e. without any edge) to make the dimensionality of the matrices match. Specifically, we denote  $N'$  as the number of added nodes. For the projection matrix  $\mathbf{U}_0$ , we can generate an additional orthogonal Gaussian random matrix  $\hat{\mathbf{U}}_0 \in \mathbb{R}^{N' \times d}$  and concatenate it with the current projection matrix  $\mathbf{U}_0$  to form the new projection matrix  $\mathbf{U}'_0$ . For other  $\mathbf{U}_i, 1 \leq i \leq q$ , we can easily find that they have all-zero elements for the empty nodes, i.e. we only need to add  $N'$  all-zero rows to match the dimensionality. Then, we can add edges to those newly added nodes using Eq. (7).

We show the framework of dynamic updating in Algorithm 3. As only local changes are involved, the updating is computationally efficient, as we specific in the following theorem.

**Theorem 3.** *The time complexity of Algorithm 3 is linear with respect to the number of changed nodes and number of changed edges respectively.*

*Proof:* See appendix. ■

Another merit of our updating method is that it has no error aggregation, i.e. the dynamic updating algorithm leads to the same results as rerunning the static algorithm. We formalize this property in the following theorem.

**Theorem 4.** *Denote  $\mathbf{A}_0$  and  $\Delta\mathbf{A}_1, \Delta\mathbf{A}_2, \dots, \Delta\mathbf{A}_t$  as the initial adjacency matrix and its dynamic changes in  $t$  time steps respectively. Denote  $\mathbf{U}$  as the final embedding results of applying Algorithm 1 to  $\mathbf{A}_0$  and then updating  $t$  times using  $\Delta\mathbf{A}_1, \Delta\mathbf{A}_2, \dots, \Delta\mathbf{A}_t$  and Algorithm 3. Denote  $\mathbf{U}'$  as the embedding results of applying Algorithm 1 to  $\mathbf{A} = \mathbf{A}_0 +$*

TABLE I  
THE STATISTICS OF DATASETS

Dataset	# Nodes	# Edges	# Labels
BlogCatalog	10,312	667,966	39
Flickr	80,513	11,799,764	47
Youtube	1,138,499	5,980,886	195
WeChat	250 million	4.8 billion	-

$\Delta \mathbf{A}_1 + \Delta \mathbf{A}_2 + \dots + \Delta \mathbf{A}_t$ . If  $\mathbf{U}$  and  $\mathbf{U}'$  are calculated using the same hyper-parameters and random seed, then  $\mathbf{U} = \mathbf{U}'$ .

*Proof:* Since the same random seed is used,  $\mathbf{U}_0 = \mathbf{U}'_0$ . Then, applying Eqs. (5) (6) (7) leads to the results. ■

Combining Theorem 3 and Theorem 4, our updating method can effectively incorporate the dynamic changes of networks with high computational efficiency.

## V. EXPERIMENTS

### A. Experimental Setting

To comprehensively evaluate the efficacy of RandNE, we first conduct experiments on 3 moderate-scale social networks<sup>4</sup>: BlogCatalog, Flickr, Youtube, and then evaluate our method on a billion-scale network, WeChat. The statistics of the datasets are summarized in Table I.

We compare our method with the following baselines:

- DeepWalk [2]<sup>5</sup> uses random walks and the skip-gram model to learn embeddings. We use two parameter settings: one suggested in the paper and one used in the implementation of the authors, and report the best results.
- LINE [5]<sup>6</sup> explicitly preserves the first two order proximities, denoted as LINE<sub>1st</sub> and LINE<sub>2nd</sub> respectively. We exclude the results of concatenating them because no obvious improvement is observed. We use the default parameter settings except the number of training samples, which we conduct a line search for the optimal value.
- Node2vec [9]<sup>7</sup> generalizes DeepWalk and LINE by using potentially biased random walks. We use the default settings for all parameters except the bias parameters  $p, q$ , which we conduct a grid search from  $\{0.5, 1, 2\}$ .
- SDNE [4]<sup>8</sup> proposes a deep auto-encoder to preserve the first and the second order proximities simultaneously. We use the default parameter settings and the auto-encoder structure in the implementation of the authors.

There are also other methods like GraRep [11] and M-NMF [13], but we exclude them here for their scalability issues. We also exclude the results of SDNE on Youtube because it fails to terminate in one week. On the WeChat network, as all these baselines cannot terminate within acceptable time, we mainly compare our method with other simpler graph-based methods.

For our method RandNE, we set the order  $q = 3$  with a grid search for the weights. Please note that the weights only

<sup>4</sup><http://socialcomputing.asu.edu/pages/datasets>

<sup>5</sup><https://github.com/phanein/deepwalk>

<sup>6</sup><https://github.com/tangjianpu/LINE>

<sup>7</sup><https://github.com/snap-stanford/snap>

<sup>8</sup><https://github.com/suanrong/SDNE>

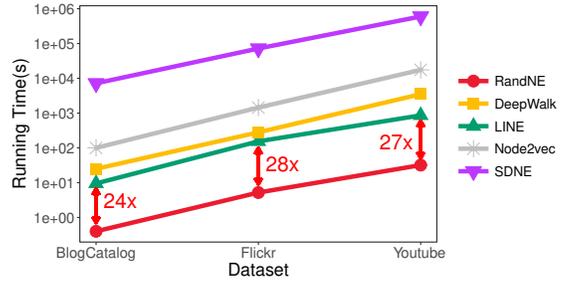


Fig. 1. The running time comparison of different methods. Our method RandNE can boost the efficiency by more than 24 times over state-of-the-art methods on all networks.

TABLE II  
AUC SCORES OF NETWORK RECONSTRUCTION

Dataset	BlogCatalog	Flickr	Youtube
RandNE	<b>0.958</b>	<b>0.953</b>	0.982
DeepWalk	0.843	0.951	0.995
LINE <sub>1st</sub>	0.901	0.947	<b>0.999</b>
LINE <sub>2nd</sub>	0.761	0.936	0.970
Node2vec	0.805	0.890	0.952
SDNE	0.950	0.919	-

affect the last step of our algorithm (i.e., line 6 in Algorithm 1), so our proposed method is very efficient in tuning them. For the node classification task, we use transition matrix to replace the adjacency matrix because substantial improvement is observed. All hyper-parameters of our method and the baselines are tuned using a small validation set, which we set as 10% for moderate-scale networks and 1% for the billion-scale network.

For all the methods, we uniformly set the dimensionality as  $d = 128$  unless stated otherwise. All experiments are conducted in a single PC with 2 I7 processors and 48GB memory, except for Section V-C, where we run our method in a distributed cluster.

### B. Moderate-scale Networks

1) *Running Time Comparison:* To compare the efficiency of different methods, we first report the running time of all the methods in Figure 1. The results show that RandNE can boost the efficiency by more than 24 times over the baselines on all networks, which is consistent to our time complexity analysis. Note that the baselines are tested using the source code published by their authors. We realize that there might be slight differences in the programming languages and implementation details, but the impact of these factors can be safely ignored considering the improvement of 24 times. So we directly report their results for reproducibility concerns. The extreme efficiency lays the foundation for applying RandNE to billion-scale networks.

2) *Network Reconstruction:* One basic objective for network embedding is to reconstruct the network. Specifically, we train embedding vectors and rank pairs of nodes according to their inner product similarities. Then, the top ranking pairs are used to reconstruct the network because large similarities

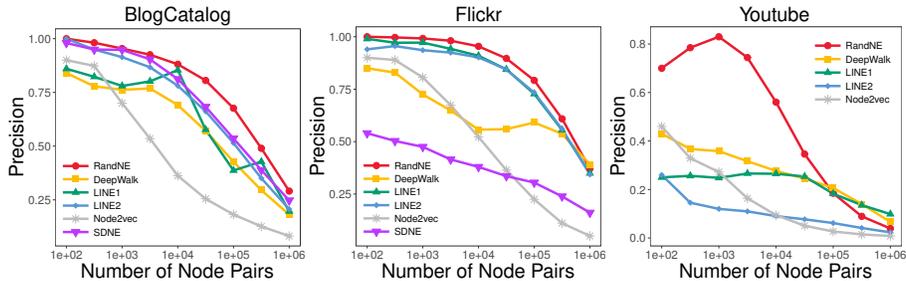


Fig. 2. The Precision@K of network reconstruction on moderate-scale networks. We train embedding vectors and rank pairs of nodes according to their inner-product similarities. The top ranking pairs are used to reconstruct the network. The results show that our proposed method can well preserve the network structure and reconstruct the given network.

indicate high probabilities of having edges. For the evaluation metrics, we use Area Under Curve (AUC) [34] and Precision@K [4] defined as:

$$Precision@K = \frac{1}{K} \sum_{i=1}^K \delta_i, \quad (8)$$

where  $\delta_i = 1$  means the  $i^{th}$  reconstructed pair is correct,  $\delta_i = 0$  represents a wrong reconstruction and  $K$  is the number of evaluated pairs. On Youtube, the number of possible pairs of nodes  $\frac{N(N-1)}{2}$  is too large to evaluate, so we sample 1% for evaluation, as in [12].

The results are shown in Table II and Figure 2. Our proposed method consistently outperforms the baselines on the metric Precision@K. On AUC, our method achieves the best performance on BlogCatalog and Flickr, and has comparable performance on Youtube. Considering the significant improvement in efficiency and the simplicity of our model, we regard the performance of RandNE in accuracy aspect to be satisfactory and somewhat beyond expectation.

3) *Link Prediction*: Link prediction, aiming to predict future links using the current network structure, is an important task of network embedding. In our experiments, we randomly hide 30% of the edges for testing. After training embedding vectors on the rest of the network, we rank pairs of nodes in a similar way as network reconstruction and evaluate the results on the testing network. The process is repeated 5 times and the average results are reported.

From Table III and Figure 3, we can see that our proposed method still outperforms the baselines in nearly all cases except the AUC score on Youtube, as in network reconstruction. The results demonstrate that besides reconstructing the network, RandNE also has good inference abilities, which we attribute to effectively preserving the high-order proximity.

4) *Node Classification*: Node classification is a typical application of network embedding. Specifically, we follow the experimental setting in baselines and randomly split the nodes into training and testing set. Then, an one-vs-all logistic regression with L2 regularization [35] is trained using the embeddings on the training set, and tested on the testing set. Following [5], we normalize each row of the embedding vectors. We use two measurements, Macro-F1 and Micro-F1

TABLE III  
AUC SCORES OF LINK PREDICTION.

Dataset	BlogCatalog	Flickr	Youtube
RandNE	<b>0.944</b>	<b>0.940</b>	0.887
DeepWalk	0.760	0.938	0.909
LINE <sub>1st</sub>	0.667	0.909	0.847
LINE <sub>2nd</sub>	0.762	0.932	<b>0.959</b>
Node2vec	0.650	0.865	0.778
SDNE	0.940	0.926	-

[2], to evaluate the performance. The average results of 5 runs are reported.

From Figure 4, different networks show different patterns in terms of node classification performance. On Flickr, RandNE achieves the best results while Node2vec and LINE show good performance on BlogCatalog and Youtube respectively. One plausible explanation for such inconsistency is that different networks have different inherent structures corresponding to the specific classification task, and no single method can dominate others on all datasets. But in general, we can safely conclude that RandNE has comparable results with the baselines in node classification while being significantly faster.

5) *Structural Role Classification*: Recently, how to preserve the structural role of nodes in network embedding attracts some research attention [36], which has important applications such as influence maximization and measuring node centrality. To validate the effectiveness of our method in structural role classification, we conduct experiments on three air-traffic networks<sup>9</sup> from Brazilian, European and American as in [36], where the networks have 131 nodes and 2,006 edges, 399 nodes and 11,986 edges, 1,190 nodes and 27,198 edges respectively. The networks are constructed by assigning airports as nodes and airlines as edges. Each node is assigned a ground-truth label ranging from 1 to 4 to indicate the level of activities of the airports.

The experimental setting is similar to node classification in Section V-B4 except that we use accuracy, i.e. the percentage of nodes whose labels are correctly predicted, as the measurement since the labels have the same size. We uniformly set the dimensionality of embedding as 16 since the networks have small sizes. The average results of 20 runs are reported.

<sup>9</sup><https://github.com/leoribeiro/struc2vec>

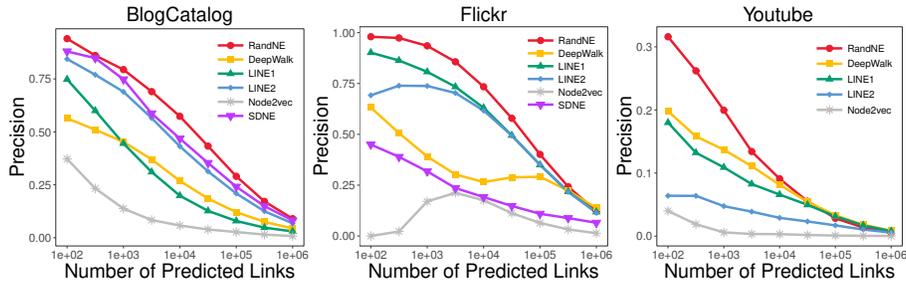


Fig. 3. The Precision@K of link prediction on moderate-scale networks. We randomly split the network into training and testing. After training embedding vectors on the training network, we predict links by ranking similarities of node pairs and make evaluation on the testing network. The results show that our proposed method outperforms the baselines in link prediction.

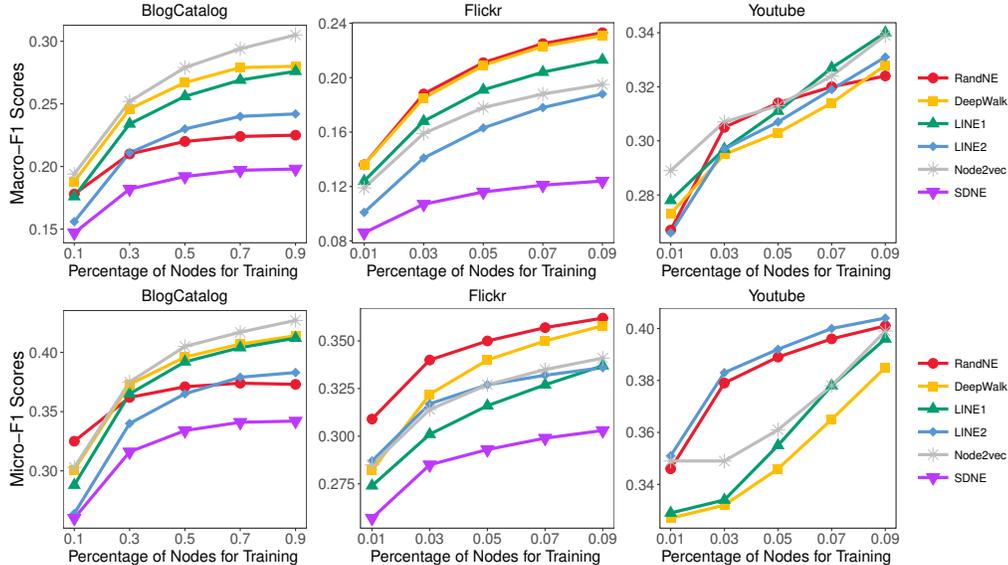


Fig. 4. The results of node classification on moderate-scale networks. We train an one-vs-all logistic regression on the embedding vectors as the classifier. The results show that our proposed method achieves comparable performance with the baselines.

From Figure 5, we can see that RandNE consistently achieves the best results on European Flights Network. On Brazilian and American Flights Network, RandNE is only second to SDNE with tiny differences. However, RandNE is much more efficient than SDNE by about 4 orders of magnitude (see Figure 1). The results demonstrate that RandNE can effectively capture the structural role of nodes.

6) *Analysis:* In RandNE, we use iterative random projection to preserve high-order proximities. Here we analyze the effect of the proximity order, or equivalently, the number of iterations  $q$ . We report the results of varying  $q$  from 1 to 3 with the same experimental settings. For brevity, we only report AUC scores of link prediction and the accuracy of structural role classification on American Flights in Figure 6, while other datasets and tasks show similar patterns. The results show that iterative random projection ( $q > 1$ ) greatly and consistently outperforms the simple random projection ( $q = 1$ ), demonstrating the importance of preserving high-order proximities in network embedding.

To verify the scalability of RandNE, we conduct experi-

ments on random networks (i.e. the Erdos Renyi model [37]). We record the running time when fixing the number of nodes (as one million) or fixing the number of edges (as ten million) while varying the other. Figure 7 shows that the running time grows linearly with respect to the number of nodes and number of edges respectively, verifying the scalability of RandNE.

Since our method is based on random projection, we also empirically analyze the impact of randomization. Specifically, we set different random seeds for RandNE with the same experimental setting, and report the mean value and standard deviation of 10 runs. For brevity, we only report the AUC scores of network reconstruction and link prediction in Table IV, while other tasks show similar patterns. The results show that our method is quite stable with respect to randomization.

### C. A Billion-scale Network

WeChat<sup>10</sup> is one of the largest online social networks in China with more than one billion active users. We use the friendships data provided by WeChat from January 21, 2011

<sup>10</sup><http://www.wechat.com/en/>

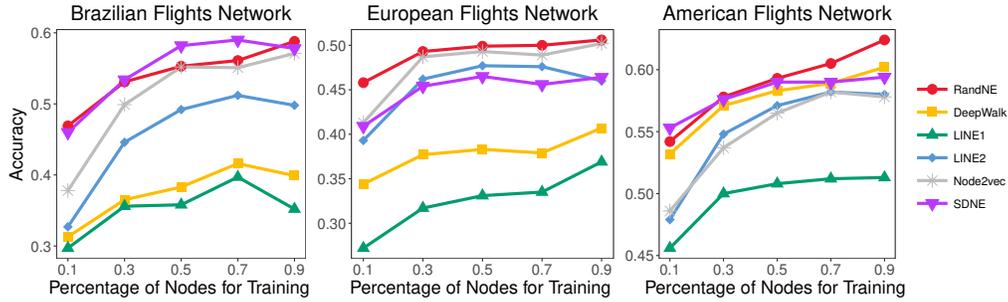


Fig. 5. The accuracy of structural role classification. We use embedding vectors to predict the structural role of nodes in air-traffic networks. The results show that RandNE effectively captures the structural role of nodes.

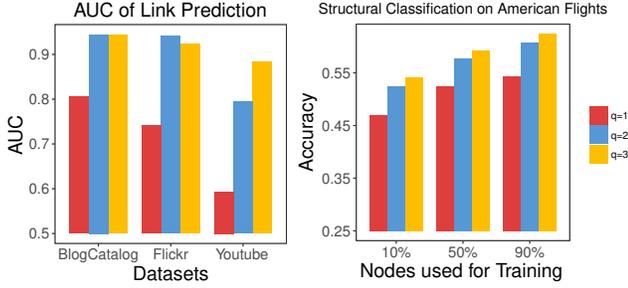


Fig. 6. Parameter analysis. Figure shows that the high-order proximity ( $q > 1$ ) greatly outperforms the simple random projection ( $q = 1$ ), demonstrating the importance of preserving high-order proximities.

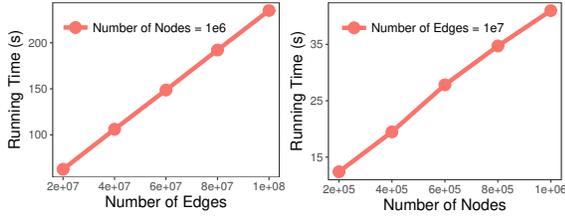


Fig. 7. Scalability analysis. Figure shows the linear time complexity of our method with respect to the number of nodes and number of edges respectively.

TABLE IV  
MEAN VALUE AND STANDARD DEVIATION OF AUC WITH 10 DIFFERENT INITIALIZATIONS.

Dataset	BlogCatalog	Flickr	Youtube
Reconstruction	$0.958 \pm 0.001$	$0.954 \pm 0.003$	$0.981 \pm 0.002$
Link Prediction	$0.945 \pm 0.001$	$0.941 \pm 0.002$	$0.887 \pm 0.001$

(the launch day of WeChat) to January 20, 2013, which contains 250 million nodes and 4.8 billion edges in total. The data is strictly anonymized for privacy purposes. Since no node label information is available, we mainly conduct experiments on network reconstruction and link prediction. As none of the aforementioned network embedding method can be applied to network of such a scale, we compare our method with two widely used graph-based measures: Common Neighbors and Adamic Adar [38]. The benchmark accuracy of random guessing is also added. The experiments are conducted in a distributed cluster with 16 computing servers, where each

TABLE V  
AUC SCORES OF NETWORK RECONSTRUCTION ON WECHAT NETWORK.

Method	AUC
RandNE	<b>0.989</b>
Common Neighbors	0.783
Adamic Adar	0.783
Random	0.500

server has 2 Xeon E5 CPU and 128GB memory. For RandNE, we set the embedding dimensionality as  $d = 512$ .

1) *Network Reconstruction*: The experimental setting is similar to moderate-scale networks (Section V-B2), i.e. we rank pairs of nodes according to their inner product similarities and reconstruct the network. We report the AUC scores in Table V. Here we omit the other metric, Precision@K, because the number of possible node pairs  $\frac{N(N-1)}{2} \approx 10^{16}$  is so large that even sampling is infeasible.

The results show that our proposed method greatly outperforms Common Neighbor and Adamic Adar. A plausible reason is that our method preserves the high-order proximity information in the embedding vectors by performing the iterative random projection, while the baselines only count local proximities. Adamic Adar, as a frequency-weighted modification of Common Neighbors, has the same accuracy as Common Neighbors because on the billion-scale network, AUC score mainly depends on whether two nodes have neighbors instead of the weights.

2) *Dynamic Link Prediction*: To simulate the evolving scenarios of real networks, we first randomly split the network into 30% training and 70% testing, train embedding vectors on the training set and evaluate the link prediction results on the testing set. Then, we randomly select 10% (with respect to the whole network) from the testing set as the evolving part and add it to the training set. After updating the embedding vectors using the new training data, we evaluate the link prediction results on the rest of the network. The process is repeated until when 70% of the network becomes training and the rest 30% is testing. We adopt two versions of our method: one by dynamic updating as the network evolves (RandNE-D), and one by re-running the algorithm at each time (RandNE-R). To fairly compare the effectiveness of our dynamic updating method and re-running the algorithm, we use the same random

TABLE VI  
AUC SCORES OF DYNAMIC LINK PREDICTION ON WECHAT.

Observed Edges	30%	40%	50%	60%	70%
RandNE-D	<b>0.646</b>	<b>0.689</b>	<b>0.726</b>	<b>0.756</b>	<b>0.780</b>
RandNE-R	<b>0.646</b>	<b>0.689</b>	<b>0.726</b>	<b>0.756</b>	<b>0.780</b>
Common Neighbors	0.575	0.611	0.647	0.681	0.712
Adamic Adar	0.575	0.611	0.647	0.681	0.712
Random	0.500	0.500	0.500	0.500	0.500

TABLE VII  
THE RUNNING TIME OF RANDNE VIA DISTRIBUTED COMPUTING.

Number of Sub-clusters	1	2	3	4
Running Time(s)	82157	46029	33965	24757

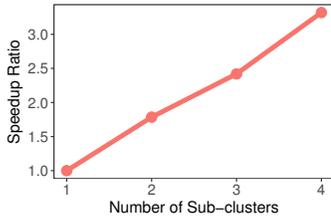


Fig. 8. The speedup ratio of RandNE via distributed computing.

seed for RandNE-D and RandNE-R.

The results in Table VI show that our proposed method again consistently outperforms the baselines on the AUC scores. Besides, RandNE-D shows identical results as RandNE-R, verifying that our dynamic updating has no error aggregation. All methods have larger AUC scores as training data increases because more information is provided.

3) *Speedup via Distributed Computing*: We evaluate the performance of our method in distributed computing by reporting the speedup ratio. We divide the 16 servers into 4 sub-clusters, with each sub-cluster containing 4 servers. We vary the number of sub-clusters used for distributed computing and record the running time and speedup ratio. Figure 8 shows that our method has a linear speedup ratio with a slope of approximately 0.8. The slope is slightly less than 1 because of subtle differences in the servers and some extra costs, e.g. reading data. We also report the exact running time in Table VII. It shows that RandNE can learn all the node embeddings of WeChat within 7 hours with 16 normal servers, which is promising for real billion-scale networks.

We also analyze the impact of the proximity order  $q$  on WeChat network, which shows similar patterns as moderate-scale networks in Figure 6. We omit the results here for brevity.

## VI. CONCLUSION

In this paper, we study the problem of embedding billion-scale networks while preserving high-order proximities. We propose RandNE, a novel and simple network embedding method based on random projection and design an iterative projection procedure to efficiently preserve the high-order proximities. Theoretical analysis shows that i) RandNE is more computationally efficient than the existing methods by orders

of magnitude, ii) it can well support distributed computing without any communication cost in the calculation, and iii) it can efficiently incorporate the dynamic changes of the networks without error aggregation. Extensive experimental results on multiple datasets with different scales demonstrate the efficiency and efficacy of our proposed method.

One future direction is to generalize this idea to incorporate node attributes and node labels. It is also interesting to explore other random projection methods beyond Gaussian projection.

## ACKNOWLEDGMENT

This work was supported in part by National Program on Key Basic Research Project (No. 2015CB352300), National Natural Science Foundation of China (No. 61772304, No. 61521002, No. 61531006, No. 61702296), National Natural Science Foundation of China Major Project (No. U1611461), the research fund of Tsinghua-Tencent Joint Laboratory for Internet Innovation Technology, and the Young Elite Scientist Sponsorship Program by CAST. All opinions, findings and conclusions in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

## REFERENCES

- [1] P. Cui, X. Wang, J. Pei, and W. Zhu, "A survey on network embedding," *IEEE Transactions on Knowledge and Data Engineering*, 2018.
- [2] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge Discovery and Data Mining*. ACM, 2014, pp. 701–710.
- [3] Z. Zhang, P. Cui, X. Wang, J. Pei, X. Yao, and W. Zhu, "Arbitrary-order proximity preserved network embedding," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2019, pp. 2778–2786.
- [4] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 1225–1234.
- [5] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th International Conference on World Wide Web*. WWW, 2015, pp. 1067–1077.
- [6] S. S. Vempala, *The random projection method*. American Mathematical Soc., 2005, vol. 65.
- [7] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.
- [8] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems*, 2013, pp. 3111–3119.
- [9] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 855–864.
- [10] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang, "Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec," in *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 2018, pp. 459–467.
- [11] S. Cao, W. Lu, and Q. Xu, "Grarep: Learning graph representations with global structural information," in *Proceedings of the 24th ACM International Conference on Information and Knowledge Management*. ACM, 2015, pp. 891–900.
- [12] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016.

[13] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, "Community preserving network embedding," in *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, 2017.

[14] S. Chen, S. Niu, L. Akoglu, J. Kovačević, and C. Faloutsos, "Fast, warped graph embedding: Unifying framework and one-click algorithm," *arXiv preprint arXiv:1702.05764*, 2017.

[15] C. Yang, M. Sun, Z. Liu, and C. Tu, "Fast network embedding enhancement via high order proximity approximation," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, 2017.

[16] K. Tu, P. Cui, X. Wang, F. Wang, and W. Zhu, "Structural deep embedding for hyper-networks," *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, 2018.

[17] D. Zhu, P. Cui, Z. Zhang, J. Pei, and W. Zhu, "High-order proximity preserved embedding for dynamic networks," *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[18] J. Li, H. Dani, X. Hu, J. Tang, Y. Chang, and H. Liu, "Attributed network embedding for learning in a dynamic environment," in *Proceedings of the 26th ACM International Conference on Information and Knowledge Management*, 2017.

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

[20] C. Zhou, Y. Liu, X. Liu, Z. Liu, and J. Gao, "Scalable graph embedding for asymmetric proximity," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, 2017, pp. 2942–2948.

[21] Y. Dong, N. V. Chawla, and A. Swami, "metapath2vec: Scalable representation learning for heterogeneous networks," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 135–144.

[22] T. Chen and Y. Sun, "Task-guided and path-augmented heterogeneous network embedding for author identification," in *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. ACM, 2017, pp. 295–304.

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

[24] C. Tu, W. Zhang, Z. Liu, and M. Sun, "Max-margin deepwalk: Discriminative learning of network representation," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, 2016.

[25] S. Pan, J. Wu, X. Zhu, C. Zhang, and Y. Wang, "Tri-party deep network representation," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, 2016, pp. 1895–1901.

[26] Z. Yang, W. Cohen, and R. Salakhudinov, "Revisiting semi-supervised learning with graph embeddings," in *International Conference on Machine Learning*, 2016, pp. 40–48.

[27] R. I. Arriaga and S. Vempala, "An algorithmic theory of learning: Robust concepts and random projection," *Machine Learning*, vol. 63, no. 2, pp. 161–182, 2006.

[28] Q. Shi, C. Shen, R. Hill, and A. Hengel, "Is margin preserved after random projection?" in *Proceedings of the 29th International Conference on Machine Learning*, 2012, pp. 591–598.

[29] K. M. Choromanski, M. Rowland, and A. Weller, "The unreasonable effectiveness of structured random orthogonal embeddings," in *Advances in Neural Information Processing Systems*, 2017, pp. 218–227.

[30] E. Liberty, "Simple and deterministic matrix sketching," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2013, pp. 581–588.

[31] C. Eckart and G. Young, "The approximation of one matrix by another of lower rank," *Psychometrika*, vol. 1, no. 3, pp. 211–218, 1936.

[32] B. Vastenhout and R. H. Bisseling, "A two-dimensional data distribution method for parallel sparse matrix-vector multiplication," *SIAM review*, vol. 47, no. 1, pp. 67–95, 2005.

[33] E. G. Boman, K. D. Devine, and S. Rajamanickam, "Scalable matrix computations on large scale-free graphs using 2d graph partitioning," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM, 2013, p. 50.

[34] T. Fawcett, "An introduction to roc analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[35] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "Liblinear: A library for large linear classification," *Journal of Machine Learning Research*, vol. 9, no. Aug, pp. 1871–1874, 2008.

[36] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo, "struc2vec: Learning node representations from structural identity," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 385–394.

[37] P. Erdos and A. Rényi, "On the evolution of random graphs," *Publ. Math. Inst. Hung. Acad. Sci.*, vol. 5, no. 1, pp. 17–60, 1960.

[38] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *Journal of the Association for Information Science and Technology*, vol. 58, no. 7, pp. 1019–1031, 2007.

[39] Z. Zhang, P. Cui, J. Pei, X. Wang, and W. Zhu, "Timers: Error-bounded svd restart on dynamic networks," in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, 2018.

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

## APPENDIX

### A. Proof of Theorem 1

*Proof:* From the Johnson-Lindenstrauss property [27], for any  $\mathbf{s} \in \mathbb{R}^N$ , we have

$$P[(1 - \epsilon) \|\mathbf{s}\|_2^2 \leq \|\mathbf{s} \cdot \mathbf{R}\|_2^2 \leq (1 + \epsilon) \|\mathbf{s}\|_2^2] \geq 1 - 2e^{-\frac{(\epsilon^2 - \epsilon^3)d}{4}}. \quad (9)$$

Then, we can get:

$$\begin{aligned} & P[\|\mathbf{S} \cdot \mathbf{S}^T - \mathbf{U} \cdot \mathbf{U}^T\|_2 > \epsilon \|\mathbf{S}^T \cdot \mathbf{S}\|_2] \\ &= P\left[\sup_{\|\mathbf{x}\|_2=1} \left| \|\mathbf{x} \cdot \mathbf{S}\|_2^2 - \|\mathbf{x} \cdot \mathbf{S} \cdot \mathbf{R}\|_2^2 \right| > \epsilon \sup_{\|\mathbf{x}\|_2=1} \|\mathbf{x} \cdot \mathbf{S}\|_2^2\right] \\ &\leq P\left[\exists \mathbf{s} \in \text{rowspan}\{\mathbf{S}\}, \left| \|\mathbf{s}\|_2^2 - \|\mathbf{s} \cdot \mathbf{R}\|_2^2 \right| > \epsilon \|\mathbf{s}\|_2^2\right] \\ &\leq r_{\mathbf{S}} P\left[\mathbf{s}, \left| \|\mathbf{s}\|_2^2 - \|\mathbf{s} \cdot \mathbf{R}\|_2^2 \right| > \epsilon \|\mathbf{s}\|_2^2\right] = 2r_{\mathbf{S}} e^{-\frac{(\epsilon^2 - \epsilon^3)d}{4}}. \end{aligned} \quad (10)$$

The last line is resulted from the union bound and Eq. (9). ■

### B. Proof of Theorem 3

*Proof:* Denote the number of changed nodes and number of changed edges as  $N'$  and  $M'$  respectively.

Then, the time complexity from line 1 to line 5 is  $O(N' \cdot d^2 + N' \cdot q \cdot d)$ , i.e. linear with respect to  $N'$ .

For line 6 to line 10, consider any edge  $(s, t)$  in the changes of edges, i.e.  $\Delta \Lambda(s, t) \neq 0$ . It is easy to see that only the embedding vectors of nodes in its  $q$ -step neighborhood, i.e. nodes that are connected to node  $s$  or node  $t$  with a path of length that is no longer than  $q$ , will change. Formally, denote the expected size of  $q$ -step neighborhood of a randomly chosen edge as  $\mathcal{R}(q)$ . Then, the expected time complexity from line 6 to line 10 is  $O(M' \cdot \mathcal{R}(q))$ , i.e. linear with respect to  $M'$ . As for  $\mathcal{R}(q)$ , it is a constant that depends on the individual network structure. Generalized from [39], the exact results for two typical types of networks are as follows:

- For random networks, (i.e. the Erdos Renyi model [37]),  $\mathcal{R}(q) \approx 2(d_{avg})^q$ , where  $d_{avg}$  is the average degree of the network.
- For Barabasi Albert model [40], a widely studied example of preferential attachment networks,  $\mathcal{R}(q) \approx 2\left[\frac{6}{\pi^2}(\log d_{max} + \gamma)\right]^q$ , where  $d_{max}$  is the maximum degree of the network and  $\gamma \approx 0.58$  is the Euler-Mascheroni constant.

As a result, the overall time complexity is  $O(N' \cdot d^2 + N' \cdot q \cdot d + M' \cdot \mathcal{R}(q))$ , which concludes the proof. ■