This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2822283, IEEE Transactions on Knowledge and Data Engineering

High-order Proximity Preserved Embedding For Dynamic Networks

Dingyuan Zhu, Peng Cui, Ziwei Zhang, Jian Pei, *Fellow, IEEE,* Wenwu Zhu, *Fellow, IEEE*

Abstract—Network embedding, aiming to embed a network into a low dimensional vector space while preserving the inherent structural properties of the network, has attracted considerable attention. However, most existing embedding methods focus on the static network while neglecting the evolving characteristic of real-world networks. Meanwhile, most of previous methods cannot well preserve the high-order proximity, which is a critical structural property of networks. These problems motivate us to seek an effective and efficient way to preserve the high-order proximity in embedding vectors when the networks evolve over time. In this paper, we propose a novel method of Dynamic High-order Proximity preserved Embedding (DHPE). Specifically, we adopt the generalized SVD (GSVD) to preserve the high-order proximity. Then, by transforming the GSVD problem to a generalized eigenvalue problem, we propose a generalized eigen perturbation to incrementally update the results of GSVD to incorporate the changes of dynamic networks. Further, we propose an accelerated solution to the DHPE model so that it achieves a linear time complexity with respect to the number of nodes and number of changed edges in the network. Our empirical experiments on one synthetic network and several real-world networks demonstrate the effectiveness and efficiency of the proposed method.

Index Terms—Dynamic Network, High-order Proximity, Network Embedding

1 Introduction

Network embedding has attracted increasing attention in recent years. The basic idea is to embed a network into a low-dimensional vector space where the proximity structure of the network is preserved so that the network analysis and prediction tasks can be conducted in the vector space. Although the state-of-the-art proposed methods are demonstrated to be effective in a variety of applications, such as link prediction [1], [2], classification [3], [4], [5] and clustering [6], [7], most of these methods are designed for static networks. In real world, however, networks are dynamic in nature, where the edges between nodes evolve over time. For example, users add or delete friends in social networks, or neurons establish new connections in brain networks. These newly added or deleted edges raise a new challenge to network embedding. Suppose that we have learned the node embeddings based on the edges appearing before time t. How to efficiently update these node embeddings at time $t+\Delta t$ so that the changed network structure caused by the newly added/deleted edges during Δt can be reflected by the updated node embeddings?

The previous methods have demonstrated that the node proximity on networks is a critical structure that should be maintained in the embedding space [8], [9]. The high-order proximity is proven to be important in capturing the structure of network [10]. However, it is much more challenging for the high-order proximity preserved network embedding methods to efficiently incorporate the newly added/deleted edges, because any changed edge will affect the high-order proximities of far more nodes than the two nodes directly involved by the edge. An extreme case is an infinite order proximity preserving method in a connected graph, where changing any edge will change the proximity of any pair of nodes in the whole graph. How to efficiently address the changed edges in high-order proximity preserved network embedding is still an open problem.

Before network embedding, calculating the high-order proximities in a large network per se is unaffordable in real applications. Ou et al. [10] recently find a general form for the commonly used high-order proximities like Katz [11], and the transformed form makes it possible to derive the embedding vectors via generalized SVD without explicitly calculating the high-order proximity matrix. But how to efficiently incorporate the newly added/deleted edges in the generalized SVD framework has not been investigated, and thus become the major obstacle of high-order proximity preserved embedding for dynamic networks.

In this paper, we propose DHPE to preserve the high-order proximity on embedding dynamic undirected networks. After transforming the GSVD problem into a generalized eigenvalue problem, we are able to incorporate the dynamically changed edges through matrix perturbation and thus derive the updated node embeddings while preserving the global high-order proximities. As the method is a global updating, it is inevitable to have some complex terms involving global structural information during matrix perturbation process, causing the efficiency bottleneck of the method. To address this, we further propose an accelerated solution scheme for these complex terms, which significantly reduces the computing complexity.

To verify the advantages of our algorithm, we conduct extensive experiments on both synthetic dataset and real-world large-scale datasets. The empirical experiments demonstrate that DHPE can approximate the high-order

- *D. Zhu, P. Cui, Z. Zhang, and W. Zhu are with the Department of Computer Science and Technology in Tsinghua University, Beijing 100084, China. E-mail: zhudy11@mails.tsinghua.edu.cn, cuip@tsinghua.edu.cn, zw-zhang16@mails.tsinghua.edu.cn, wwzhu@tsinghua.edu.cn*
- *J. Pei is with the School of Computing Science, Simon Fraser University. E-mail: jpei@cs.sfu.ca*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2822283, IEEE Transactions on Knowledge and Data Engineering

proximities well during the update process and significantly outperforms the baseline methods in several tasks, including link prediction, node recommendation and multi-label classification. It is also worthwhile to mention that the proposed method reaches a linear time complexity with respect to the number of nodes in the network. Considering that the complexity of globally updating network embedding is at least linear with respect to the number of nodes in the network, our method is in the same order with the theoretical lower bound in the scalability.

The main contributions of this paper are as follows:

- We investigate the important and challenging problem of high-order proximity preserved embedding on dynamic networks for the first time.
- We propose a GSVD-based method for dynamic network embedding with an incremental updating based on matrix perturbation.
- We propose an acceleration scheme for the method and optimize its computing complexity to be in the same order with the theoretical lower bound.
- We comprehensively evaluate the effectiveness and efficiency of DHPE on several synthetic and large-scale real-world networks in various applications.

The rest of the paper is organized as follows. In Section 2, we review the related work. In Section 3, we formally define the problem of dynamic network embedding and introduce the framework of DHPE. We introduce the acceleration of the algorithm in Section 4 and report the experimental results in Section 5. We conclude the paper in Section 6.

2 Related Work

The research on network embedding algorithm can be traced back to the time when graph embedding algorithms [12], [13], [14] have been proposed. The graph embedding algorithms aim to preserve the feature similarity in the embedded latent space. These algorithms build an adjacency matrix of graph which are constructed from the feature similarity, and then embed the graph to a low-dimensional vector space [15]. For example, Isomap [13] aims to find the low-dimensional representations for a data set by approximately preserving the geodesic distances between data pairs. These graph embedding works focus on modeling the observed first-order relationship (i.e. edges in graph) between vertexes, which means they may overfit the original graph. These algorithms have drawbacks in real-world network, because the network inference ability is seriously limited in such an embedding space.

Motivated by the graph embedding techniques, Hoff et. al. [16] first proposed to learn latent space representation of nodes for network analysis, and they apply it to link prediction problem [17]. Handcock et. al. [18] proposed to apply the latent space approach to clustering in graph. And Zhu et. al. [19] proposed to address the classification problem in network with graph embedding model. Because of the popularity of networked data, network embedding has received more and more attention in recent years. Deepwalk [9] uses the language modeling techniques and learn the latent representations of a network by truncated random walks. LINE [8] embeds the network into a low-dimensional space where the first-order and second-order proximity between nodes is preserved. Node2vec [20] learn a mapping of nodes to a low-dimensional space of features that maximizes the likelihood of preserving network neighborhoods of nodes. GraRep [21] tries to model the high-order proximity between nodes in network, but the time cost of computing high-order proximity is too high. HOPE [10] proposes a high-order proximity preserved embedding method. Some other network embedding methods are proposed to process heterogeneous networks [4], [22], [23] or networks with content information [2], [24].

All the aforementioned approaches can only handle static networks. Some approaches based on first-order proximity can be transformed into dynamic models straightforwardly. For example, the SVD of adjacency matrix can be used as a simple way to get the embedding of network. Tong et. al. [25] proposed a fast eigen-tracking algorithm, which can be used to update the solution of the SVD problem when the matrix is symmetric. Thus we may get a simple dynamic model base on SVD, but it only preserves the first-order proximity between nodes. It is still not clear how to design a high-order proximity preserved embedding method when the networks are evolving over time.

3 The DHPE Method

In this section, we formally define the problem of dynamic network embedding with high-order proximity preserved and introduce our method, DHPE.

3.1 Notation and Definition

We first summarize some notations and definitions used in this paper. A dynamic network at time step t is defined as $\mathbf{G}^{(t)} = \{\mathbf{V}^{(t)}, \mathbf{E}^{(t)}\}$, where $\mathbf{V}^{(t)} = \{v_1^{(t)}, v_2^{(t)}, ..., v_N^{(t)}\}$ denotes a set of nodes and N is the number of the nodes. $\mathbf{E}^{(t)}$ is the set of edges between the nodes. In this paper, we mainly consider undirected networks, so edges in $\mathbf{E}^{(t)}$ are undirected. The adjacency matrix is denoted as $\mathbf{A}^{(t)}$, and $\Delta\mathbf{A}$ denotes the change of adjacency matrix. We define $\mathbf{U}^{(t)} \in \mathcal{R}^{N\times d}$ as the embedding matrix of the network $\mathbf{G}^{(t)}$, where the i-th row, $\mathbf{u}_i^{(t)}$,is the embedding vector of $v_i^{(t)}$ and d is the embedding dimension. Similar to previous network embedding settings, d is a preset constant and $d << N$. Let $\mathbf{S}^{(t)}$ denotes the high-order proximity matrix of the network $\mathbf{G}^{(t)}$, where $\mathbf{S}_{ij}^{(t)}$ is the proximity between $v_i^{(t)}$ and $v_j^{(t)}$.

In this paper, we focus on the problem of dynamic network embedding with high-order proximity preserved. At each time step, nodes and edges may be added/deleted. By treating the added/deleted nodes as isolated nodes, all the changes in the network can be regarded as the changes of the edges [25]. For the ease of presentation, we consider the number of nodes as constant.

The problem of dynamic network embedding can be split into two parts. First, we build a static model to embed the static network to a low-dimensional vector space, where high-order proximities between nodes are preserved. Second, we propose a dynamic model to update the embedding of nodes in the network at following time steps. These two parts are summarized as follow:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2822283, IEEE Transactions on Knowledge and Data Engineering

Problem 1. *Static network embedding: given adjacency matrix $\mathbf{A}^{(t)}$ at time step t; output the embedding matrix $\mathbf{U}^{(t)}$ using static model.*

Problem 2. *Dynamic network embedding: given adjacency matrix $\{\mathbf{A}^{(t+1)}, \mathbf{A}^{(t+2)}, ..., \mathbf{A}^{(t+i)}\}$ at time steps $t+1, t+2, ..., t+i$ and the embedding matrix $\mathbf{U}^{(t)}$ at time step t; output the embedding matrix $\{\mathbf{U}^{(t+1)}, \mathbf{U}^{(t+2)}, ..., \mathbf{U}^{(t+i)}\}$ at time step $t+1, t+2, ..., t+i$.*

3.2 GSVD-based Static Model

We begin with the model of static network embedding for preserving high-order proximity. Specifically, as proposed in [10], we aim to preserve high-order proximity in the embedding matrix with the following objective function:

$$\min ||\mathbf{S} - \mathbf{U}\mathbf{U}'^{\top}||_F^2 \tag{1}$$

,where $\mathbf{U}, \mathbf{U}' \in \mathcal{R}^{N\times d}$. In matrix decomposition, $\mathbf{U}$ and $\mathbf{U}'$ can be seen as the basis and the coordinate [26]. For undirected networks, $\mathbf{U}$ and $\mathbf{U}'$ are highly correlated, as shown later in equation (3), and without loss of generality we choose $\mathbf{U}$ as the embedding matrix.

We choose Katz Index [11] as $\mathbf{S}$ because it is one of the most widely used measures of high-order proximity. It can be formulated as:

$$\begin{aligned} \mathbf{S}^{Katz} &= {\mathbf{M}_a}^{-1}\mathbf{M}_b \\ \mathbf{M}_a &= (\mathbf{I} - \beta\mathbf{A}) \\ \mathbf{M}_b &= \beta\mathbf{A} \end{aligned} \tag{2}$$

, where β is a decay parameter and $\mathbf{I}$ is the identity matrix. β determines how fast the weight of a path decays when the length of path grows. It should be properly set to preserve the series converging, and we discuss how to set β in Section 5.1.

Then, as in [10], the original objective function can be solved by the generalized SVD (GSVD) method [26]. By GSVD method, we can derive the singular values and singular vectors of $\mathbf{S}$ without knowing $\mathbf{S}$. Formally, the optimal embedding vectors of objective function (1) can be given as:

$$\begin{aligned} \mathbf{U} &= [\sqrt{\sigma_1}\mathbf{v}_1^l, ..., \sqrt{\sigma_d}\mathbf{v}_d^l] \\ \mathbf{U}' &= [\sqrt{\sigma_1}\mathbf{v}_1^r, ..., \sqrt{\sigma_d}\mathbf{v}_d^r] \end{aligned} \tag{3}$$

,where $\{\sigma_1, \sigma_2, ..., \sigma_N\}$ is the singular values of $\mathbf{S}$ sorted in decreasing order. $\mathbf{v}_i^l$ and $\mathbf{v}_i^r$ are corresponding left and right singular vectors of σ_i. When $\mathbf{S}$ is symmetric, $|\mathbf{v}_i^l| = |\mathbf{v}_i^r|$, where $|\cdot|$ means taking absolute value element-wisely. According to [10], the error bound of the static method is:

$$||\mathbf{S} - \mathbf{U}\mathbf{U}'^{\top}||_F^2 = \sum_{i=d+1}^{N} \sigma_i^2 \tag{4}$$

3.3 Problem Transformation for Dynamic Model

The problem of the dynamic modeling is that given $\Delta\mathbf{A}$ and $\mathbf{U}^{(t)}$, how to incrementally update $\mathbf{U}^{(t)}$ to $\mathbf{U}^{(t+1)}$. With equation (3), the embedding matrix $\mathbf{U}^{(t)}$ only depends on the singular values and singular vectors of $\mathbf{S}^{(t)}$, so we focus on how to update them. At time step t, the results of GSVD satisfy the following equations:

$$\begin{aligned} \mathbf{S}^{(t)} &= {\mathbf{M}_a^{(t)}}^{-1}\mathbf{M}_b^{(t)} = \mathbf{V}^{l(t)}\mathbf{\Sigma}^{(t)}\mathbf{V}^{r(t)\top} \\ \mathbf{\Sigma}^{(t)} &= diag({\sigma_1}^{(t)}, {\sigma_2}^{(t)}, ..., {\sigma_N}^{(t)}) \end{aligned} \tag{5}$$

where $\mathbf{V}^{l(t)}$ and $\mathbf{V}^{r(t)}$ are singular vectors in matrices (e.g. $\mathbf{v}_i^{l(t)}$ is the i-th row of $\mathbf{V}^{l(t)}$). As mentioned before, the problem reduces to developing an efficient way to update the singular values ($\mathbf{\Sigma}^{(t)}$) and singular vectors ($\mathbf{V}^{l(t)}$,$\mathbf{V}^{r(t)}$) to $\mathbf{\Sigma}^{(t+1)}$, $\mathbf{V}^{l(t+1)}$ and $\mathbf{V}^{r(t+1)}$ respectively. Due to high time complexity of calculating $\mathbf{S}^{(t+1)}$, it is technically difficult to directly utilize the equation (5) for the updating.

Here we propose to transform the GSVD problem into generalized eigenvalue problem, so that the incremental updating is feasible. In undirected networks, the adjacency matrix $\mathbf{A}$ and the high-order proximity matrix $\mathbf{S}$ are symmetric matrices. From [27], GSVD can be transformed into the generalized eigenvalue problem:

$${\mathbf{M}_a}^{-1}\mathbf{M}_b\mathbf{X} = \mathbf{\Lambda}\mathbf{X} \tag{6}$$

$$\mathbf{\Lambda} = diag(\lambda_1, \lambda_2, ..., \lambda_N) \tag{7}$$

$$\lambda_i = \sigma_i \cdot sgn(\mathbf{v}_i^l \cdot \mathbf{v}_i^r) \tag{8}$$

$$\mathbf{X} = \mathbf{V}^l \tag{9}$$

,where $\{\lambda_i\}$ are the eigenvalues of $\mathbf{S}$ in descending order, and $\mathbf{X}$ is a matrix which contains the corresponding eigenvectors of λ_i and $sgn()$ is the Sign function. By multiplying the matrix $\mathbf{M}_a$ on both sides of equation (6), we have:

$$\mathbf{M}_b\mathbf{X} = \mathbf{M}_a\mathbf{\Lambda}\mathbf{X} \tag{10}$$

The above formula is exactly the formulation of generalized eigenvalue problem. And the results of the generalized eigenvalue problem can also be transformed back into the results of GSVD problem:

$$\begin{aligned} \mathbf{v}_i^l &= \mathbf{x}_i \\ \sigma_i &= |\lambda_i| \\ \mathbf{v}_i^r &= \mathbf{x}_i \cdot sgn(\lambda_i) \end{aligned} \tag{11}$$

,where $\mathbf{x}_i$ is the i-th column of the matrix $\mathbf{X}$, which represents the corresponding eigenvectors of λ_i.

Based on above equations, we can conveniently derive $\mathbf{\Lambda}$ and $\mathbf{X}$ from $\mathbf{\Sigma}$, $\mathbf{V}^l$ and $\mathbf{V}^r$, and vice versa. That means if we have $\mathbf{\Sigma}^{(t)}$, $\mathbf{V}^{l(t)}$ and $\mathbf{V}^{r(t)}$, we can get $\mathbf{X}^{(t)}$ and $\mathbf{\Lambda}^{(t)}$ according to equation (8) and (9). Meanwhile, if we have $\mathbf{X}^{(t+1)}$ and $\mathbf{\Lambda}^{(t+1)}$, we can get $\mathbf{\Sigma}^{(t+1)}$, $\mathbf{V}^{l(t+1)}$ and $\mathbf{V}^{r(t+1)}$ according to equation (11). Then the key problem is how to efficiently update $\mathbf{X}^{(t)}$ to $\mathbf{X}^{(t+1)}$. In next subsection, we propose generalized eigen perturbation to fulfill this task.

3.4 Generalized Eigen Perturbation

The goal of generalized eigen perturbation is to update $\mathbf{X}^{(t)}$ to $\mathbf{X}^{(t+1)}$. As the perturbation process for any time step t is the same, we omit the (t) superscript for brevity. Specifically, given the change of adjacency matrix $\Delta\mathbf{A}$ between two consecutive time steps, the change of $\mathbf{M}_a$ and $\mathbf{M}_b$ can be represented as:

$$\Delta\mathbf{M}_a = -\beta\Delta\mathbf{A},\ and\ \Delta\mathbf{M}_b = \beta\Delta\mathbf{A} \tag{12}$$

We use $\Delta\mathbf{\Lambda}$ and $\Delta\mathbf{X}$ to denote the change of the eigenvalues and eigenvectors. With the equation (10), we have the following:

$$(\mathbf{M}_b+\Delta\mathbf{M}_b)(\mathbf{X}+\Delta\mathbf{X}) = (\mathbf{M}_a+\Delta\mathbf{M}_a)(\mathbf{\Lambda}+\Delta\mathbf{\Lambda})(\mathbf{X}+\Delta\mathbf{X}) \tag{13}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2822283, IEEE Transactions on Knowledge and Data Engineering

For a specific eigen-pair, we have:

$$(\mathbf{M}_b+\Delta\mathbf{M}_b)(\mathbf{x}_i+\Delta\mathbf{x}_i) = (\lambda_i+\Delta\lambda_i)(\mathbf{M}_a+\Delta\mathbf{M}_a)(\mathbf{x}_i+\Delta\mathbf{x}_i) \tag{14}$$

First, we introduce the calculation of $\Delta\lambda_i$. By expanding equation (14) and using the fact $\mathbf{M}_b x_i = \lambda_i \mathbf{M}_a \mathbf{x}_i$, we get:

$$\begin{aligned}&\mathbf{M}_b\Delta\mathbf{x}_i + \Delta\mathbf{M}_b x_i + \Delta\mathbf{M}_b\Delta\mathbf{x}_i \\ &= \lambda_i\mathbf{M}_a\Delta\mathbf{x}_i + \lambda_i\Delta\mathbf{M}_a\mathbf{x}_i + \lambda_i\Delta\mathbf{M}_a\Delta\mathbf{x}_i \\ &+ \Delta\lambda_i\mathbf{M}_a\mathbf{x}_i + \Delta\lambda_i\mathbf{M}_a\Delta\mathbf{x}_i + \Delta\lambda_i\Delta\mathbf{M}_a\mathbf{x}_i + \Delta\lambda_i\Delta\mathbf{M}_a\Delta\mathbf{x}_i\end{aligned} \tag{15}$$

The higher order terms such as $\Delta\mathbf{M}_b\Delta\mathbf{x}_i$ and $\lambda_i\Delta\mathbf{M}_a\Delta\mathbf{x}_i$ can be removed as they have limited effects on the accuracy of the solution [28]. Removing the higher order terms in equation (15) and left multiplying both sides by $\mathbf{x}_i^\top$, we have:

$$\begin{aligned}&\mathbf{x}_i^\top\mathbf{M}_b\Delta\mathbf{x}_i + \mathbf{x}_i^\top\Delta\mathbf{M}_b\mathbf{x}_i = \lambda_i\mathbf{x}_i^\top\mathbf{M}_a\Delta\mathbf{x}_i \\ &+ \lambda_i\mathbf{x}_i^\top\Delta\mathbf{M}_a\mathbf{x}_i + \Delta\lambda_i\mathbf{x}_i^\top\mathbf{M}_a\mathbf{x}_i\end{aligned} \tag{16}$$

Because $\mathbf{M}_a$ and $\mathbf{M}_b$ are symmetric, we have:

$$\mathbf{x}_i^\top\mathbf{M}_b = \lambda_i\mathbf{x}_i^\top\mathbf{M}_a \tag{17}$$

Using the above equation, $\mathbf{x}_i^\top\mathbf{M}_b\Delta\mathbf{x}_i$ and $\lambda_i\mathbf{x}_i^\top\mathbf{M}_a\Delta\mathbf{x}_i$ can be cancelled from both sides of Equation (16), we get:

$$\mathbf{x}_i^\top\Delta\mathbf{M}_b\mathbf{x}_i = \lambda_i\mathbf{x}_i^\top\Delta\mathbf{M}_a\mathbf{x}_i + \Delta\lambda_i\mathbf{x}_i^\top\mathbf{M}_a\mathbf{x}_i$$

After a few manipulations, we have the formula for calculating the change of the eigenvalue λ_i:

$$\Delta\lambda_i = \frac{\mathbf{x}_i^\top\Delta\mathbf{M}_b\mathbf{x}_i - \lambda_i\mathbf{x}_i^\top\Delta\mathbf{M}_a\mathbf{x}_i}{\mathbf{x}_i^\top\mathbf{M}_a\mathbf{x}_i} \tag{18}$$

For ease of presentation, we define some notations that will be used:

$$\begin{aligned}\mathbf{H}_a(i,j) &= \mathbf{x}_i{}^\top\Delta\mathbf{M}_a\mathbf{x}_j \quad (i,j=1..d) \\ \mathbf{H}_b(i,j) &= \mathbf{x}_i{}^\top\Delta\mathbf{M}_b\mathbf{x}_j \quad (i,j=1..d) \\ \mathbf{F}_a(i,j) &= \mathbf{x}_i{}^\top\mathbf{M}_a\mathbf{x}_j \quad (i,j=1..d) \\ \mathbf{F}_b(i,j) &= \mathbf{x}_i{}^\top\mathbf{M}_b\mathbf{x}_j \quad (i,j=1..d)\end{aligned} \tag{19}$$

Equation (18) can be rewritten as:

$$\Delta\lambda_i = \frac{\mathbf{H}_b(i,i) - \lambda_i\mathbf{H}_a(i,i)}{\mathbf{F}_a(i,i)} \tag{20}$$

Next, we introduce the calculation of $\Delta\mathbf{x}_i$. Between two consecutive time steps, the evolution of the network is usually smooth. With the matrix perturbation theory [29], we can assume that the change of the eigenvectors $\Delta\mathbf{x}_i$ is the linear expression of the top-d eigenvectors:

$$\Delta\mathbf{x}_i = \sum_{j=1,j\neq i}^{d}\alpha_{ij}\mathbf{x}_j \tag{21}$$

,where α_{ij} is the coefficient indicating the contribution of $\mathbf{x}_j$ to $\Delta\mathbf{x}_i$. Considering equation (15), replacing all $\Delta\mathbf{x}_i$ terms with equation (21) and multiplying the term $\mathbf{x}_p{}^\top$ $(for\, 1 \leq p \leq d, p \neq i)$ on both size, we can get $d-1$ equations:

$$\begin{aligned}&\mathbf{H}_b(p,i) + \sum_{j=1,j\neq i}^{d}\mathbf{H}_b(p,j)\alpha_{ij} + \sum_{j=1,j\neq i}^{d}\mathbf{F}_b(p,j)\alpha_{ij} \\ &= (\lambda_i+\Delta\lambda_i)\mathbf{H}_a(p,i) + (\lambda_i+\Delta\lambda_i)\sum_{j=1,j\neq i}^{d}\mathbf{H}_a(p,j)\alpha_{ij} \\ &+ \Delta\lambda_i\mathbf{F}_a(p,i) + (\lambda_i+\Delta\lambda_i)\sum_{j=1,j\neq i}^{d}\mathbf{F}_a(p,j)\alpha_{ij}\end{aligned} \tag{22}$$

For a specific $\Delta\mathbf{x}_i$, we have $d-1$ equations $(for\, 1 \leq p \leq d, p \neq i)$ and $d-1$ unknowns $(\alpha_{ij}, for\, 1 \leq j \leq d, j \neq i)$. Let $\alpha_i = [\alpha_{i1}, ..., \alpha_{i(i-1)}, \alpha_{i(i+1)}, ..., \alpha_{id}]^\top$, $\mathbf{W} \in \mathcal{R}^{(d-1)\times(d-1)}$ and $\mathbf{B} \in \mathcal{R}^{(d-1)\times 1}$ are the coefficient matrices. α_i can be obtained by the following formulas:

$$\mathbf{B}(p) = \mathbf{H}_b(p,i) - (\lambda_i+\Delta\lambda_i)\mathbf{H}_a(p,i) - \Delta\lambda_i\mathbf{F}_a(p,i) \tag{23}$$

$$\begin{aligned}\mathbf{W}(p,j) &= (\lambda_i+\Delta\lambda_i)\sum_{j=1,j\neq i}^{d}\mathbf{H}_a(p,j) - \sum_{j=1,j\neq i}^{d}\mathbf{H}_b(p,j) \\ &+ (\lambda_i+\Delta\lambda_i)\sum_{j=1,j\neq i}^{d}\mathbf{F}_a(p,j) - \sum_{j=1,j\neq i}^{d}\mathbf{F}_b(p,j)\end{aligned} \tag{24}$$

$$\alpha_i = \mathbf{W}^{-1}\mathbf{B} \tag{25}$$

With the equations (20),(21)and(25), we can calculate the change of the eigenvalues and eigenvectors. Then, we calculate $\mathbf{\Lambda}^{(t+1)}$ and $\mathbf{X}^{(t+1)}$ by adding $\Delta\lambda_i$ and $\Delta\mathbf{x}_i$ to the corresponding eigenvalues and eigenvectors respectively. In order to maintain the consistency of the generalized eigenvalue problem, we normalize the eigenvector $\mathbf{X}^{(t+1)}$ at the end of the updating process.

4 Acceleration and Complexity Analysis

In this section we introduce the acceleration of the algorithm and we provide a complexity analysis of the proposed framework.

4.1 Algorithm Acceleration

For calculating the change of the eigenvalues and eigenvectors, calculating $\mathbf{F}_a$ and $\mathbf{F}_b$ is necessary. However, it is inefficient to recalculate $\mathbf{F}_a$ and $\mathbf{F}_b$ at each time step, as these terms involving global structural information.

Considering the definition of $\mathbf{F}_a$ in equation (19), the term $\mathbf{x}_i{}^\top\mathbf{M}_a\mathbf{x}_j$ can be expanded as:

$$\mathbf{x}_i{}^\top\mathbf{M}_a\mathbf{x}_j = \sum_{<l,r>\in\mathbf{E}}\mathbf{M}_a(l,r)\mathbf{x}_{li}\mathbf{x}_{rj} \tag{26}$$

With above equation, the time complexity of computing $\mathbf{F}_a{}^{(t)}$ is $O(Md^2)$, where M is number of edges in $\mathbf{E}^{(t)}$. Likewise, the time complexity of computing $\mathbf{F}_b{}^{(t)}$ is the same as that of $\mathbf{F}_a{}^{(t)}$. Calculating these terms causes the efficiency bottleneck of our dynamic model.

To address this problem, we propose an incremental calculation scheme for these complex terms. Specifically,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2822283, IEEE Transactions on Knowledge and Data Engineering

Algorithm 1 Dynamic High-order Proximity preserved Embedding

Input: $\mathbf{U}^{(t)}$, $\mathbf{\Sigma}^{(t)}$, $\mathbf{V}^{l(t)}$, $\mathbf{V}^{r(t)}$, ${\mathbf{F}_a}^{(t)}$, ${\mathbf{F}_b}^{(t)}$,change of the adjacency matrix $\Delta\mathbf{A}$ between time t and t+1
Output: $\mathbf{U}^{(t+1)}$, $\mathbf{\Sigma}^{(t+1)}$, $\mathbf{V}^{l(t+1)}$, $\mathbf{V}^{r(t+1)}$, ${\mathbf{F}_a}^{(t+1)}$, ${\mathbf{F}_b}^{(t+1)}$
Calculate $\mathbf{\Lambda}^{(t)}$, $\mathbf{X}^{(t)}$ by equation (8) and (9)
Calculate $\Delta\mathbf{M}_a$, $\Delta\mathbf{M}_b$ by equation (12)
Calculate $\mathbf{H}_a$ and $\mathbf{H}_b$ by equation (19)
for i = 1 to d **do**
  Calculate $\Delta\lambda_i$ by equation (20)
  $\lambda_i^{(t+1)} = \lambda_i^{(t)} + \Delta\lambda_i$
  Calculate $\mathbf{B}$ and $\mathbf{W}$ by equation (23) and (24)
  Calculate α_i by equation (25)
  Calculate $\Delta\mathbf{x}_i$ by equation (21)
  $\mathbf{x}_i^{(t+1)} = \mathbf{x}_i^{(t)} + \Delta\mathbf{x}_i$
end for
Normalize $\mathbf{X}^{(t+1)}$ and update α_{ij}
Update ${\mathbf{F}_a}^{(t+1)}$ and ${\mathbf{F}_b}^{(t+1)}$ by equation (30) and (31)
Calculate $\mathbf{\Sigma}^{(t+1)}$, $\mathbf{V}^{l(t+1)}$, $\mathbf{V}^{r(t+1)}$ by equation (11)
Calculate $\mathbf{U}^{(t+1)}$ by equation (3)

by using the expression of the change of the eigenvectors $\Delta\mathbf{x}_i$, we can efficiently update $\mathbf{F}_a$ and $\mathbf{F}_b$ at new time step. Without loss of generality, we use updating $\mathbf{F}_a^{(t)}$ as an example:

$$ {\mathbf{F}_a}^{(t+1)}(i,j) = (\mathbf{x}_i^{(t)} + \Delta\mathbf{x}_i)^\top ({\mathbf{M}_a}^{(t)} + \Delta\mathbf{M}_a)(\mathbf{x}_j^{(t)} + \Delta\mathbf{x}_j) \tag{27} $$

Replacing $\Delta\mathbf{x}_i$ and $\Delta\mathbf{x}_j$ with equation (21), we get:

$$ {\mathbf{F}_a}^{(t+1)}(i,j) = \sum_{l=1}^{d} \gamma_{il} {\mathbf{x}_l^{(t)}}^\top \cdot ({\mathbf{M}_a}^{(t)} + \Delta\mathbf{M}_a) \cdot \sum_{r=1}^{d} \gamma_{jr}\mathbf{x}_r^{(t)} \tag{28} $$

,where

$$ \gamma_{ij} = \begin{cases} 1 & i = j, \\ \alpha_{ij} & i \neq j. \end{cases} \tag{29} $$

Replacing all ${\mathbf{x}_i^{(t)}}^\top {\mathbf{M}_a}^{(t)} \mathbf{x}_r^{(t)}$ terms with ${\mathbf{F}_a}^{(t)}(l,r)$ and replacing all ${\mathbf{x}_l^{(t)}}^\top \Delta\mathbf{M}_a\mathbf{x}_r^{(t)}$ terms with ${\mathbf{H}_a}^{(t)}(l,r)$, we get:

$$ {\mathbf{F}_a}^{(t+1)}(i,j) = \sum_{l=1}^{d}\sum_{r=1}^{d} \gamma_{il}\gamma_{jr}({\mathbf{F}_a}^{(t)}(l,r) + {\mathbf{H}_a}^{(t)}(l,r)) \tag{30} $$

With above equation, the time complexity of updating ${F_a}^{(t+1)}$ will reduce to $O(d^4)$. Likewise, ${F_b}^{(t+1)}$ can be updated in the same fashion:

$$ {\mathbf{F}_b}^{(t+1)}(i,j) = \sum_{l=1}^{d}\sum_{r=1}^{d} \gamma_{il}\gamma_{jr}({\mathbf{F}_b}^{(t)}(l,r) + {\mathbf{H}_b}^{(t)}(l,r)) \tag{31} $$

With this acceleration technique, the time complexity of our model is unrelated to the number of edges in the network at last time step, which greatly improves the efficiency of the algorithm. Algorithm 1 lists the steps of our method.

4.2 Complexity Analysis

Here, we analyze the complexity of the proposed framework. According to [26], the time complexity of the static model is $O(Md^2L)$, where M is the number of edges in the network at time step t and L is the iteration number. This is the partial update algorithm, and we only need to run the static model once at the beginning of the update algorithm. At time step t, ${\mathbf{F}_a}^{(t)}$ and ${\mathbf{F}_b}^{(t)}$ need to be calculated by definition which takes $O(Md^2)$.

The efficiency of dynamic model is summarized in Lemma (4.1). The time complexity of dynamic model is linear with respect to the number of the nodes in the network and total number of the time steps.

Lemma 4.1. ***Complexity of dynamic model*** *Suppose T is the total number of the time steps, s is the average number of edges in ΔA. The time complexity of dynamic model is $O(T((N + s)d^2 + d^4))$, the space complexity of dynamic model is $O(Nd + d^2 + s)$*

Proof. We use step (i) to refer the i^{th} step in the Algorithm 1. For time complexity, step (1) takes $O(Nd)$ and step (2) takes $O(s)$. The term ${\mathbf{x}_i}^\top \Delta\mathbf{M}_a\mathbf{x}_j$ can be expanded as:

$$ {\mathbf{x}_i}^\top \Delta\mathbf{M}_a\mathbf{x}_j = \sum_{<l,r>\in\Delta\mathbf{A}} \Delta\mathbf{M}_a(l,r)\mathbf{x}_{li}\mathbf{x}_{rj} \tag{32} $$

With above equation, the time complexity of computing $\mathbf{H}_a$ and $\mathbf{H}_b$ in step (3) is $O(sd^2)$. Steps (5)(6) takes $O(1)$ and step (7) takes $O(d^3)$. Then step (8) takes $O(d^3)$ and step (9) takes $O(Nd)$. Updating $\mathbf{x}_i^{(t+1)}$ in step (10) takes $O(N)$. Therefore, the updating process from step (4) to (11) takes $O(Nd^2 + d^4)$. Normalizing $\mathbf{X}^{(t+1)}$ in step (12) takes $O(Nd)$. Finally, step (13) takes $O(d^4)$ and step(14)(15) takes $O(Nd)$. Thus the overall time complexity for T time steps is $O(T((N + s)d^2 + d^4))$.

For space complexity, it takes $O(Nd)$ to store $\mathbf{U}^{(t)}$, $\mathbf{\Sigma}^{(t)}$, $\mathbf{V}^{l(t)}$ and $\mathbf{V}^{r(t)}$ for each time step. In step (1), it takes $O(Nd)$ and $O(d)$ to store and calculate $\mathbf{X}^{(t)}$ and $\mathbf{\Lambda}^{(t)}$ respectively. In step (2), it takes $O(s)$ to store and calculate $\Delta\mathbf{M}_a$ and $\Delta\mathbf{M}_b$. In step (3), it takes $O(d^2)$ to store and calculate $\mathbf{H}_a$ and $\mathbf{H}_b$. From step (4) to (11), the space cost of coefficient matrix and α_i is $O(d^2)$. In step (13), it takes $O(d^2)$ to store and update ${\mathbf{F}_a}^{(t+1)}$ and ${\mathbf{F}_b}^{(t+1)}$. The space cost can be reused in each time step. Thus the overall space complexity for T time steps is $O(Nd + d^2 + s)$. □

In addition, the time complexity of any method to update the embedding of all nodes in the network is at least $O(Nd + s)$, because it takes $O(s)$ deal with the changed edges, and the d-dimensional embedding vectors of each node should be updated. In large-scale networks, we have $N >> d^2$, thus the time complexity of our algorithm is in the same order with the theoretical lower bound.

5 EXPERIMENTS

In this section, we empirically evaluate the effectiveness and efficiency of the DHPE method. In particular, we evaluate the following tasks: (1) the effectiveness of preserving high-order proximity for undirected network embedding; (2) the effectiveness of our proposed model DHPE on dynamic networks; (3) the efficiency of the proposed DHPE. We first introduce the experiment setting before presenting details of the experiments.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2822283, IEEE Transactions on Knowledge and Data Engineering

5.1 Experiment Setting

5.1.1 Baseline Methods

- LINE [8]: This algorithm preserves the first-order and second-order proximity between nodes. We use LINE1 to represent LINE preserving first-order proximity and LINE2 to represent LINE preserving second-order proximity. For brevity, we exclude the results of concatenating them (i.e. LINE1 + LINE2) because it shows similar performances as the former two.
- DeepWalk [9]: This algorithm learns embedding by simulating uniform random walks. It assumes a pair of nodes similar if they are close in the random walks.
- node2vec [20]: This algorithm learns embedding by generating potentially biased random walks. Compared to DeepWalk, it has a more flexible strategy to explore neighborhoods.
- GraRep [21]: This algorithm generates node representations by explicitly computing successive powers of the random walk transition matrix, and uses the SVD to reduce their dimensionality.
- TRIP [25]: TRIP is an online algorithm to track the eigen-functions of a dynamic graph. We use the SVD method to get the embedding of static network, and apply this algorithm to update embedding incrementally.

In all experiments, we uniformly set the embedding dimension d to 100 as used in previous embedding methods. The parameter analysis of β is given in [10], and following their work, we set β as $0.8/r$, where r is spectral radius of adjacency matrix. For the baseline methods, we set the parameters by grid search. In the evaluation of link prediction and multi-label classification, the process is repeated 5 times and the average results are reported.

5.1.2 Evaluation Metrics

In the experiments, we adopt RMSE (Root Mean Square Error), Precision@k and MAP (Mean Average Precision) [10] as the evaluation metrics.

RMSE is used to evaluate the approximation error of the proximity by our updating algorithm. The formula of RMSE in our problem is:

$$RMSE = \sqrt{\frac{\|\mathbf{S} - \mathbf{U}\mathbf{U}'^{\top}\|_F^2}{N^2}}$$

Precision@k is used to evaluate the performance of link prediction, which measures the prediction precision of top k edges. The formula of Precision@k is:

$$Precision@k = \frac{|\{(i,j)|(i,j) \in \mathbf{E}_p \cap \mathbf{E}_o\}|}{|\mathbf{E}_p|}$$

where $\mathbf{E}_p$ is the set of predicted top k edges, $\mathbf{E}_o$ is the set of observed edges and $|\cdot|$ represents the size of set.

Mean Average Precision(MAP) is used to evaluate the performance of node recommendation, which measures the rank accuracy of recommended node list. The formula of MAP@k is:

$$AP@k(i) = \frac{\sum_{j=1}^{k} Precision@j(i) \cdot \delta_i(j)}{\sum_{j=1}^{k} \delta_i(j)}$$
$$MAP@k = \frac{\sum_{v_i \in \mathbf{V}} AP@k(i)}{|\mathbf{V}|}$$

where $Precision@j(i)$ is Precision@j for node v_i, and $\delta_i(j) = 1$ indicates that v_i and v_j have an edge.

5.2 Effectiveness of the Static Model

In [10], the authors demonstrate that the static model can handle asymmetric transitivity well in directed networks. However, the quality of such embedding method is not verified on undirected networks, where asymmetric transitivity does not exist. To demonstrate the importance of high-order proximities for undirected network embedding, we evaluate the effectiveness of the static model by link prediction experiment. In link prediction, we are given a network with a certain fraction of edges removed, and then we predict these missing edges. We generate datasets by randomly separating the original network into training network and testing network, where training network contains 80% edges and testing network contains the rest edges. We train the embedding vectors on training network, and evaluate the prediction performance on testing network. This experiment is conducted on the following datasets:

- BlogCatalog [30]: This is a network of social relationships of the bloggers listed on the BlogCatalog website. The network has $10,312$ nodes and $333,983$ edges.
- Catster[1]: This network contains family links between cats and cats, cats and dogs, as well as dogs and dogs from the social websites catster.com and dogster.com. The network has $623,766$ nodes and $15,699,276$ edges.
- Youtube1[2]: This is an undirected network of Youtube users and their connections. The network has $1,134,890$ nodes and $2,987,624$ edges.

As the number of possible pairs of nodes $N(N-1)$ is too large in Catster and Youtube1, we randomly sample about 0.1% pairs of nodes for evaluation as done in [10]. Then, we rank them according to the inner product between embedding vectors, and evaluate the prediction precision in top k pairs of nodes.

Figure 1 shows the precision@k of link prediction with different k. The static model, GSVD, outperforms the baselines significantly. Both GSVD and GraRep preserve the high-order proximity between nodes, and they achieve better performance in BlogCatalog dataset. But GraRep is not scalable to large-scale networks, we exclude the results of GraRep on Catster and Youtube1 datasets. The experiment results demonstrate the high-order proximity between nodes is helpful for capturing the structure of network.

1. http://konect.uni-koblenz.de/networks/petster-carnivore
2. http://konect.uni-koblenz.de/networks/com-youtube

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2822283, IEEE Transactions on Knowledge and Data Engineering

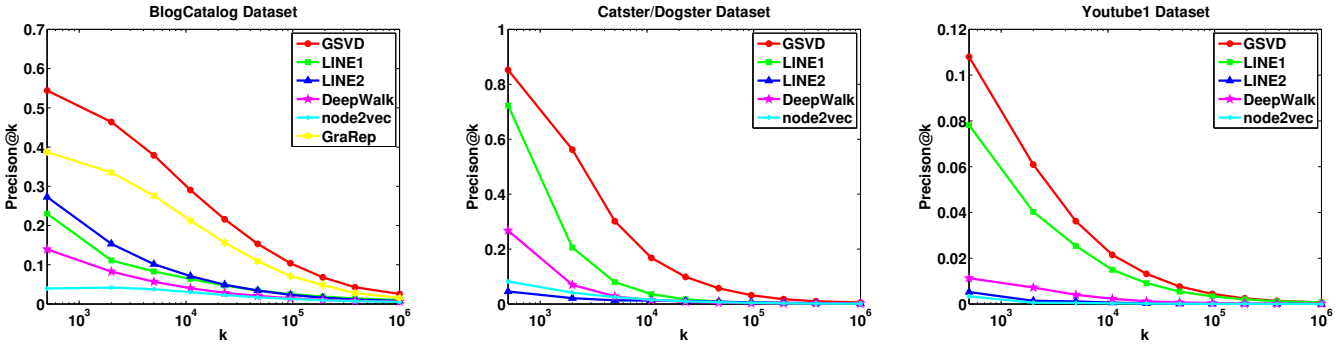


Fig. 1: Link Prediction on static networks.

5.3 Effectiveness of the Dynamic Model

In this section, we evaluate the effectiveness of the proposed dynamic model DHPE. We use TRIP and GSVD as baselines to demonstrate the necessity of dynamic updating embeddings while preserving high-order proximities in dynamic updating. For the ease of presentation, we also use "Others-best" to denote the best results of other aforementioned network embedding methods, i.e., DeepWalk, LINE-1, LINE-2, Node2vec and GraRep. As they can not handle dynamic networks, we only apply them to the static part of the network, and use the learned embeddings for subsequent tasks in the dynamic part. Meanwhile, we use GSVD-retain to retrain the embedding vectors on the entire networks as the upper bound for our method.

5.3.1 High-order Proximity Approximation

The goal of our algorithm is to preserve the high-order proximity between nodes. The error of approximation can be used to evaluate how well we preserve the high-order proximity between nodes by our updating algorithm. As the time complexity of computing the high-order proximity(Katz) is $O(N^3)$, it can not be calculated on large-scale datasets. Thus, we evaluate the approximation error on some relatively small datasets:

- Synthetic Data: We generate the synthetic data by the forest fire model [31]. The model can generate networks with power law properties. The network has 1000 nodes and 29887 edges
- Infectious[3]: This undirected network describes the face-to-face behavior of people during the exhibition. Nodes represent exhibition visitors and edges represent face-to-face contacts between the visitors. The network has 410 nodes and $17,298$ edges.

For synthetic dataset, we add a random timestamp to each edge. We separate the original network into training network and growing network by timestamps. Meanwhile, we ensure the training network is connected. The edges of the growing network is divided into 10 time slices with equal time interval. First, we use the static model to train the embedding vectors on training network denoted as GSVD-static. Then, we retrain the embedding vectors by the static model at each time step denoted as GSVD-retrain. Meanwhile, we update the embedding vectors by DHPE at each time step. Here, we compare the embeddings by DHPE with the embeddings by GSVD-static and the embeddings by GSVD-retrain. We use RMSE to evaluate high-order proximity approximation error of different methods.

3. http://konect.uni-koblenz.de/networks/sociopatterns-infectious

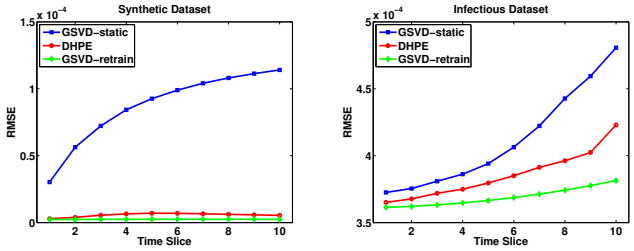


Fig. 2: High-order Proximity Approximation

Figure 2 shows the result of this experiment, and we can see that the embedding by DHPE achieves much lower RMSE than the embedding by GSVD-static. In addition, in Synthetic dataset, DHPE gets comparable results with GSVD-retrain, while having a much lower time complexity. This shows that our algorithm can effectively capture the change of high-order proximity.

5.3.2 Link Prediction

TABLE 1: Statistics of datasets used in 5.3.2. $|V|$ denotes the number of nodes , $|E_{static}|$,$|E_{grow}|$ and $|E_{test}|$ denote the number of edges in the static network, growing network and testing network respectively.

	Math	Internet	Youtube2
$\|V\|$	13,586	32,077	1,021,043
$\|E_{static}\|$	116,408	111,644	1,913,723
$\|E_{grow}\|$	41,522	58,882	590,561
$\|E_{test}\|$	96,886	137,388	835,202

In link prediction, the original network is divided into three parts according to the timestamp. The first part is the static network. The second part is the growing network, and dynamic algorithms update the embeddings through this part of the network. The final part is the testing network, where we evaluate the prediction performance. We use static methods such as GSVD and SVD to initialize the embedding in the static network. The edges of the growing network are divided into 10 time steps by timestamp. At each time step, we update the embedding by dynamic methods such as DHPE and TRIP. Finally, we use the embedding to predict

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2822283, IEEE Transactions on Knowledge and Data Engineering

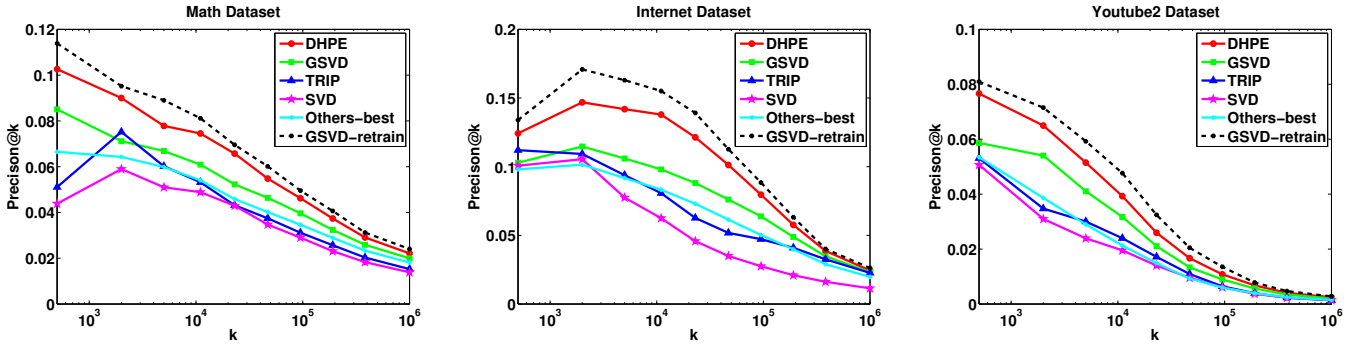


Fig. 3: Link Prediction on dynamic networks. Others-best represent GraRep in Math and Internet datasets and LINE1 in Youtube2 dataset, where GraRep and LINE1 achieve the best performance in all baselines for static networks, respectively.

TABLE 2: MAP of node recommendation on datasets. For each node, the recommended node list is ranked according to the predicted proximity between nodes. For embedding algorithms, we calculate the predicted proximity by performing inner product between embedding vectors. Others-best represent GraRep in Math and Internet datasets and LINE1 in Youtube2 dataset, where GraRep and LINE1 achieve the best performance in all baselines for static networks, respectively.

Method	Math			Internet			Youtube2		
	MAP@10	MAP@50	MAP@100	MAP@10	MAP@50	MAP@100	MAP@10	MAP@50	MAP@100
DHPE	**0.1748**	**0.1344**	**0.1126**	**0.1043**	**0.0514**	**0.0355**	**0.0840**	**0.0471**	**0.0333**
GSVD	0.1402	0.1041	0.0877	0.0713	0.0360	0.0265	0.0729	0.0419	0.0290
TRIP	0.1424	0.0988	0.0815	0.0757	0.0317	0.0215	0.0698	0.0386	0.0287
SVD	0.1295	0.0934	0.0774	0.0670	0.0266	0.0182	0.0652	0.0353	0.0262
Others-best	0.1320	0.0997	0.0831	0.0695	0.0324	0.0221	0.0686	0.0378	0.0283
GSVD-retrain	*0.1859*	*0.1432*	*0.1195*	*0.1170*	*0.0623*	*0.0414*	*0.0907*	*0.0518*	*0.0356*

the edges in testing network. Three real-world networks are used for experimental evaluation:

- Math[4]: This is a temporal network of interactions on the stack exchange web site Math Overflow.
- Internet[5]: This is the network of connections between autonomous systems of the Internet.
- Youtube2[6]: This is the social network of YouTube users and their friendship connections.

The statistics of the three networks are summarized in Table 1.

As the number of possible pairs of nodes is too large in Youtube2, we randomly sample about 0.5% pairs of nodes for evaluation as used in [10]. Then, we rank them according to the inner product between embedding vectors, and evaluate the prediction precision in top k pairs of nodes. Figure 3 shows the precision@k of link prediction on different real-world dynamic networks. Our algorithm consistently improves the link prediction accuracy on the testing networks. For example, in Internet dataset, DHPE achieves 40% improvement in precision@k, when k is equal to 10^4.

5.3.3 Node Recommendation

The setting of training procedure in this experiment is the same as link prediction. We evaluate the performance of algorithms from the node view. In orther words, we use embeddings to select the set of nodes that are most likely to connect with a particular node. Specifically, we randomly select 1000 nodes with incremental edges in testing network. For each node v_i, we derive the top 100 nodes with the highest proximity with v_i as the candidates that v_i will possible add edge to. After that, we use MAP@10, MAP@50 and MAP@100 to evaluate the quality of recommendation. We summarize our results for node recommendation in Table 2. We can see that DHPE outperforms all the baselines. In some datasets like Math and Internet, our method improves MAP by approximately 30%.

5.3.4 Multi-label Classification

As we did not find a dataset with both label information on nodes and timestamps on edges, we choose two static datasets with labels that have been used in previous work to conduct this experiment:

- Flickr [32]: This is a network of the contacts between users of the photo sharing website. The network has $80,513$ nodes, $5,899,882$ edges and 195 different labels.
- Youtube3 [33]: This is a social network between users of the popular video sharing website. The network has $1,138,499$ nodes, $2,990,443$ edges and 47 different labels.

For each network, we add a random timestamp to each edge and sort the edges by timestamp. The multi-label classification experiment settings are similar to [9], but the original network is divided into two parts according to the

4. http://snap.stanford.edu/data/sx-mathoverflow.html
5. http://konect.uni-koblenz.de/networks/topology
6. http://konect.uni-koblenz.de/networks/youtube-u-growth

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2822283, IEEE Transactions on Knowledge and Data Engineering

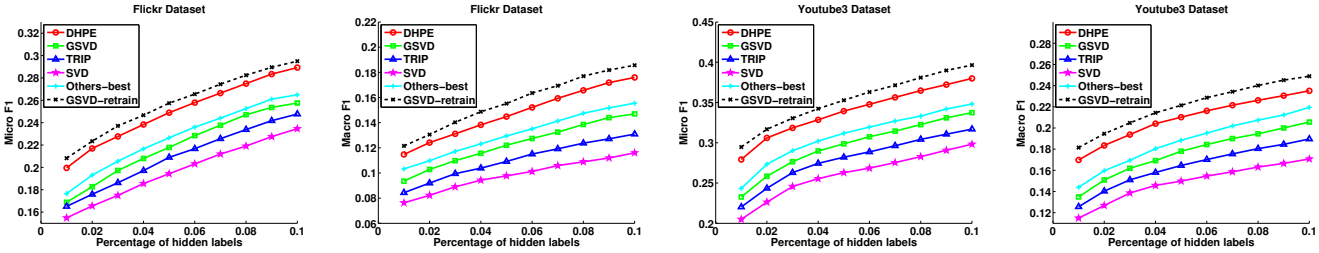
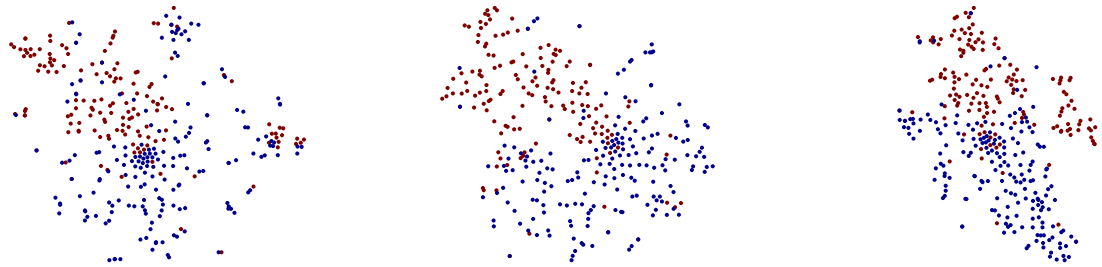


Fig. 4: Multi-label classification results. Others-best represent node2vec in Flickr and Youtube3 datasets, where node2vec achieve the best performance in all baselines for static networks.



(a) DHPE with 10% **edges** (b) DHPE with 15% **edges** (c) DHPE with 20% **edges**

Fig. 5: Visualization of network embedding.

timestamp. We use 50% of the edges as the static network and 50% of the edges as the growing network. We use static methods to initialize the embedding in the static network and update the embedding by dynamic methods in the growing network.

Then, we randomly sample a portion of the labeled nodes to predict the labels of the rest nodes. For all methods, we use a one-vs-rest logistic regression implemented by LibLinear [34] for classification. We use the Micro-F_1 and Macro-F_1 scores to evaluate the results. We compare performance while varying the percentages of hidden labels from 1% to 10%. From the results shown in Figure 4, we can see DHPE can achieve a better classification performance than baselines even if the labelled data is limited. Such an advantage is meaningful for real-world applications, because the labelled data in real-world network is usually scarce. From these experiments, we can conclude that high-order proximity and incorporating dynamic changes are both of paramount importance in network embedding, and our method shows superior performance than baselines.

5.3.5 Visualization

Visualization is another important application for network embedding. We randomly choose a subset of Youtube3, and then we generate visualizations of the network on a two-dimensional space. For the network, we add a random timestamp to each edge and sort the edges by timestamp. We use 10% of the edges as the static network and 10% of the edges as the growing network. We use static methods to initialize the embedding in the static network. The edges of the growing network are divided into 2 time steps by timestamp. At each timestep, we update the embedding by DHPE method. Then, we use the network embedding learned by DHPE as the input to the visualization tool t-SNE [35]. For nodes with different labels, we use different colors on the corresponded points. For simplicity, we randomly select two labels as a showcase. From the visualization figure shown in Figure 5, we can see DHPE capture the change of network structure caused by the newly added edges effectively.

5.3.6 Parameter Sensitivity

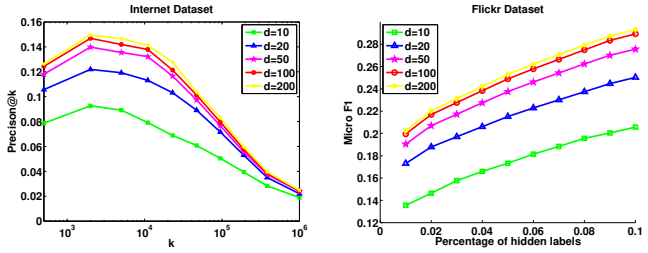
In this section, we investigate the parameter sensitivity. More specifically, we evaluate how different numbers of the embedding dimensions can affect the results of link prediction and multi-label classification. Following the previous experiment settings, we only change the numbers of the embedding dimensions to show how the dimensionality affects the performance of DHPE.

We report Precision@k on the dataset of Internet and Micro-F1 scores on the dataset of Flickr. The experiment results are shown in Figure 6. We can see that initially the performance raises when the number of dimension increases. However, when the number of dimensions continuously increases, the performance tends to be stable. This is because most of the useful information is already encoded into the embeddings. Additional dimensions consume more computing resources, but have less effect on performance. Overall, it is important to determine the appropriate number of dimensions for the latent space. When the number of dimensions is not too small, DHPE is not very sensitive to this parameter.

5.4 Efficiency of the Dynamic Model

In this section, we evaluate the efficiency of the algorithm from two aspects. First, we compare DHPE to retraining the static method to calculate the speedup ratio. Second, we

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2822283, IEEE Transactions on Knowledge and Data Engineering



(a) The Precision@k wrt the dimensionality (b) The Micro-F1 scores wrt the dimensionality

Fig. 6: Results of parameter sensitivity.

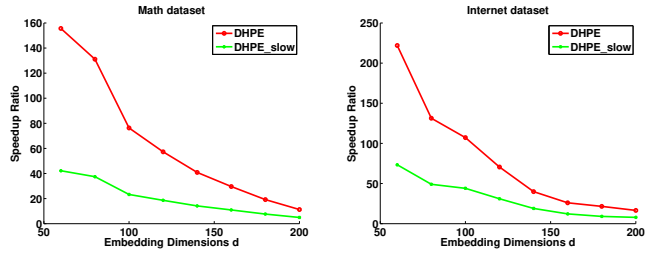


Fig. 7: The speedup ratio of DHPE wrt dimensionality.

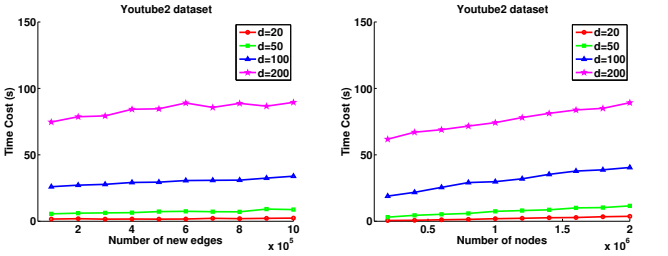
count the running time of DHPE on a real world large-scale network.

5.4.1 Speedup Ratio

We compare the efficiency of DHPE and GSVD on different datasets. At each time step, we count the time cost on the updating by DHPE and the retraining by GSVD respectively. Figure 7 shows the average speedup ratio with respect to different embedding dimensions. DHPE_slow represents the DPHE without the acceleration of Section 4, which calculates the F_a and F_b by definition. We can see that the acceleration effect is significant. The time complexity of our algorithm is unrelated to the existing edges of the network. When the network is larger, the advantage of our algorithm is more obvious. In internet dataset, we see that DHPE can achieve more than 100X speedup ratio when d is smaller than 100. As the embedding dimension d increases, the speedup ratio decreases. This is consistent with our analysis of the time complexity.

5.4.2 Scalability

We count the actual running time of DHPE in Youtube2, the largest network in our experiments. We run the experiment in a machine with 4 processors Intel Xeon 2.6GHz with 256GB of RAM. Figure 8 shows the running time of DHPE with respect to the number of new edges and the number of nodes respectively. In Figure 8a, we set the number of nodes to $1,000,000$ and vary the number of new edges from $100,000$ to $1,000,000$. In Figure 8b, we set the number of new edges to $200,000$ and vary the number of nodes from $200,000$ to $2,000,000$. When the embedding dimension d is 200, our algorithm only spends less than 100 seconds to update the embedding in the network of one million nodes.



(a) The running time wrt the number of new edges (b) The running time wrt the number of nodes

Fig. 8: The running time of DHPE in large-scale network.

6 CONCLUSION

In this paper, we study the problem of dynamic network embedding while preserving high-order proximity. We propose a scalable network embedding algorithm, called Dynamic High-Order Proximity preserved Embedding (DHPE). The algorithm preserves the high-order proximity between nodes and updates the embedding of network effectively and efficiently. With the acceleration of the algorithm, our algorithm achieves linear time complexity with respect to the number of nodes and number of changed edges in the network. The empirical study demonstrates the superiority of high-order proximities and our proposed algorithm, DHPE. Our future direction is to develop a nonlinear model to better capture the structure of dynamic networks.

REFERENCES

[1] 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.

[2] J. Chen, Q. Zhang, and X. Huang, "Incorporate group information to enhance network embedding," in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, 2016, pp. 1901–1904.

[3] Z. Yang, W. W. Cohen, and R. Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," *arXiv preprint arXiv:1603.08861*, 2016.

[4] Z. Huang and N. Mamoulis, "Heterogeneous information network embedding for meta path based proximity," *arXiv preprint arXiv:1701.05291*, 2017.

[5] 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.

[6] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, "Community preserving network embedding," 2017.

[7] F. Nie, W. Zhu, and X. Li, "Unsupervised large graph embedding." in *AAAI*, 2017, pp. 2422–2428.

[8] 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*. ACM, 2015, pp. 1067–1077.

[9] 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.

[10] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," in *Proc. of ACM SIGKDD*, 2016, pp. 1105–1114.

[11] L. Katz, "A new status index derived from sociometric analysis," *Psychometrika*, vol. 18, no. 1, pp. 39–43, 1953.

[12] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[13] J. B. Tenenbaum, V. De Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[14] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering," in *NIPS*, vol. 14, no. 14, 2001, pp. 585–591.

[15] S. Yan, D. Xu, B. Zhang, H.-J. Zhang, Q. Yang, and S. Lin, "Graph embedding and extensions: a general framework for dimensionality reduction," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 29, no. 1, pp. 40–51, 2007.

[16] P. D. Hoff, A. E. Raftery, and M. S. Handcock, "Latent space approaches to social network analysis," *Journal of the american Statistical association*, vol. 97, no. 460, pp. 1090–1098, 2002.

[17] P. D. Hoff, "Multiplicative latent factor models for description and prediction of social networks," *Computational and Mathematical Organization Theory*, vol. 15, no. 4, pp. 261–272, 2009.

[18] M. S. Handcock, A. E. Raftery, and J. M. Tantrum, "Model-based clustering for social networks," *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, vol. 170, no. 2, pp. 301–354, 2007.

[19] S. Zhu, K. Yu, Y. Chi, and Y. Gong, "Combining content and link for classification using matrix factorization," in *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2007, pp. 487–494.

[20] 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.

[21] S. Cao, W. Lu, and Q. Xu, "Grarep: Learning graph representations with global structural information," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, 2015, pp. 891–900.

[22] T. Chen and Y. Sun, "Task-guided and path-augmented heterogeneous network embedding for author identification," *arXiv preprint arXiv:1612.02814*, 2016.

[23] Y. Chen and C. Wang, "Hine: Heterogeneous information network embedding," in *International Conference on Database Systems for Advanced Applications*. Springer, 2017, pp. 180–195.

[24] X. Sun, J. Guo, X. Ding, and T. Liu, "A general framework for content-enhanced network representation learning," *arXiv preprint arXiv:1610.02906*, 2016.

[25] C. Chen and H. Tong, "Fast eigen-functions tracking on dynamic graphs," in *Proceedings of the 2015 SIAM International Conference on Data Mining*. SIAM, 2015, pp. 559–567.

[26] C. C. Paige and M. A. Saunders, "Towards a generalized singular value decomposition," *SIAM Journal on Numerical Analysis*, vol. 18, no. 3, pp. 398–405, 1981.

[27] G. Strang, G. Strang, G. Strang, and G. Strang, *Introduction to linear algebra*. Wellesley-Cambridge Press Wellesley, MA, 1993, vol. 3.

[28] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU Press, 2012, vol. 3.

[29] G. W. Stewart and J.-G. Sun, "Matrix perturbation theory (computer science and scientific computing)," 1990.

[30] Z. Reza and L. Huan, "Social computing data repository," 2009.

[31] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: densification laws, shrinking diameters and possible explanations," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM, 2005, pp. 177–187.

[32] L. Tang and H. Liu, "Relational learning via latent social dimensions," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009, pp. 817–826.

[33] ——, "Scalable learning of collective behavior based on sparse social dimensions," in *Proceedings of the 18th ACM conference on Information and knowledge management*. ACM, 2009, pp. 1107–1116.

[34] 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.

[35] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

PLACE PHOTO HERE

Dingyuan Zhu Biography text here.

PLACE PHOTO HERE

Peng Cui Biography text here.

PLACE PHOTO HERE

Ziwei Zhang Biography text here.

PLACE PHOTO HERE

Jian Pei Biography text here.

PLACE PHOTO HERE

Wenwu Zhu Biography text here.