Dynamic Network Embedding via Temporal Path Adjacency Matrix Factorization

Zhuoming Li
Southeast University
Nanjing, China
leezhuoming@seu.edu.cn

Darong Lai
School of Computer Science and Engineering, Southeast University
Key Laboratory of Computer Network and Information Integration(Southeast University), Ministry of Education
Nanjing, China
daronglai@seu.edu.cn

ABSTRACT

Network embedding has been widely investigated to learn low dimensional nodes representation of networks, and serves for many downstream machine learning tasks. Previous network embedding studies mainly focus on static networks, and cannot adapt well to the characteristics of dynamic networks which are evolving over time. Some works on dynamic network embedding have tried to improve the computation efficiency for incremental updates of embedding vectors, while others have made efforts to utilize temporal information to enhance the quality of embedding vectors. However, few existing works can fulfill both efficiency and quality requirements. In this article, a novel dynamic network embedding model named **TPANE** (**T**emporal **P**ath **A**djacency Matrix based **N**etwork **E**mbedding) is proposed. It employs a new network proximity measure: Temporal Path Adjacency, which is capable of capturing the temporal dependency between edges as well as being incrementally computed in an efficient way. It evaluates the similarity between nodes via the count of temporal paths between them, rather than making random sampling approximation, and adopts matrix factorization to obtain embedding vectors. Link prediction experiments on various real-world dynamic networks have been conducted to show the superior performance of TPANE against other state-of-the-art methods. Time consumption analysis also shows that TPANE is more efficient in incremental updates.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **Information systems** → Data mining; • **Theory of computation** → *Online learning algorithms*; **Dynamic graph algorithms**.

KEYWORDS

network embedding, dynamic network, matrix factorization, link prediction, incremental update

CIKM '22, October 17–21, 2022, Atlanta, GA, USA

ACM ISBN 978-1-4503-9236-5/22/10…$15.00
https://doi.org/10.1145/3511808.3557302

ACM Reference Format:
Zhuoming Li and Darong Lai. 2022. Dynamic Network Embedding via Temporal Path Adjacency Matrix Factorization. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM '22), October 17–21, 2022, Atlanta, GA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3511808.3557302

1 INTRODUCTION

Nowadays, much complex information is organized in the form of networks, e.g., social networks, scientific citations, and collaboration networks. Network embedding, also known as network representation learning, has been studied widely [8, 10, 13] to help relieve the difficulty of downstream machine learning tasks effectively dealing with networks. It aims to learn low-dimensional representation for each node in a network.

Previous studies for network embedding mainly focused on static networks. In recent years, the study of dynamic network embedding has addressed increased attention in recent years [38, 39], since many real-world networks are naturally dynamic-evolving over time. Dynamic network embedding has two dominant features which distinguish it from static network embedding: 1) it is temporally attributed, i.e., each edge in a dynamic network is assigned with a timestamp indicating when it is created; 2) it is expected to an online method, i.e., to produce the latest network embedding vectors as the dynamic network evolves over time. Therefore, the issue of effectiveness and of incremental updating efficiency have been two important topics of dynamic network embedding.

In the case of efficiency of dynamic network embedding, recent works [18, 36, 44] have made efforts in incremental updates with Matrix Perturbation Theory [33]. These methods obtained amazing efficiency, nearly linear time complexity with respect to the number of newly added edges, for incremental updates. However, these methods still rely on similarity measures for static network [19], including Katz Index matrix and Laplacian matrix which are derived from adjacency matrix. Such similarity measures can only learn topological information from network, leaving the abundant temporal information unused. Experiments showed [18, 36, 44] that the effectiveness of these models are inferior to their corresponding static models.

In the case of effectiveness of dynamic network embedding, Nguyen et al. have proposed CTDNE [22] to incorporate both topological information and temporal information into embedding vectors via learning the temporal dependency on temporal walks. Sufficient comparison experiments [20, 22, 32] have shown that

learning the temporal dependency is beneficial for dynamic network embedding by enhancing the quality of generated embedding vectors.

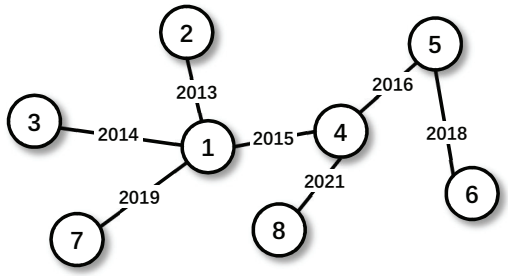

Figure 1: An example of a dynamic network. Each edge is annotated with a timestamp denoting when the edge was created.

However, the similarity based on temporal walks can hardly be incremental updated. For example, Figure 1 shows a co-author network whose nodes indicate authors and edges indicate the collaboration between authors. The timestamp on edge represents the time of collaboration. In CTDNE, the temporal transition probability from v_1 to v_6 equals to $1/3$ at time step 2018. However, after the addition of edge (v_4, v_8) at time step 2021, the transition probability from v_1 to v_6 is reduced to $1/6$ due to the decreasing of transition probability from v_4 to v_5, since there is temporal path between from v_1 to v_6 and the path is adjacent to the added edge. And such updates also occur between nodes v_1 and v_5, v_2 and v_6, v_3 and v_5 and so on. The example indicates that a large number of similarities between nodes need to be re-computed even if only one edge is added to the dynamic network. Therefore, the model is required to memorize all the generated walks and to update them incrementally while online learning. Although algorithms [20, 30] have been proposed to reduce unnecessary updates, the computation complexity is still high.

In consideration of the great success of matrix factorization approaches in promoting efficiency for incremental updates, as well as the inspiration of improving effectiveness by learning temporal dependency from CTDNE, it is hoped to find a way to obtain both these two advantages.

In this article, a novel dynamic network embedding model named **TPANE** (**T**emporal **P**ath **A**djacency Matrix based **N**etwork **E**mbedding) is proposed. It employs a new network proximity measure: Temporal Path Adjacency, which evaluates the similarity between nodes via the count of temporal paths between them. The proposed TPANE model can both produce embedding vectors with high quality via learning the temporal dependency, and do incremental updates with high efficiency when the dynamic network is evolving. It generalizes GraRep [6] from static network embedding to dynamic network embedding. The work flow of TPANE is shown in Figure 2.

In detail, the model firstly counts the temporal paths between nodes by maintaining and updating the k-step Temporal Path Adjacency (TPA) matrices according to the snapshots of a dynamic network. Then, it utilizes TPA matrices to construct the positive Pointwise Mutual Information (positive PMI) matrices. The k-step embedding vectors of the network are obtained by processing Singular Value Decomposition (SVD) on positive PMI matrices. Finally, the targeted embedding vectors are obtained by concatenating k-step embedding vectors together. Since new TPA matrices can be derived from previous TPA matrices via sparse matrix multiplication, TPANE is advantageous in the efficiency of incremental updates.

In summary, the main contributions of this article are as follows:

- A novel dynamic network embedding model TPANE is proposed, which produces embedding vectors with high quality, and is efficient for incremental updates of embedding vectors.
- A new temporal similarity measure TPA is proposed, which can captures simultaneously topological and temporal information of a dynamic network, and is suitable for incremental computation.
- Experiments on various dynamic networks and comparisons to other baseline network embedding methods are conducted to verify the effectiveness and efficiency of TPANE.

2 RELATED WORKS

Previous network embedding methods mainly focused on learning topological information of static networks. Laplacian Eigenmaps [3] preserved the local neighborhood proximity by computing the Eigenvectors of the Graph Laplacian. Deepwalk [25] proposed the first skip-gram based framework for network embedding by treating nodes as words and generating random walks as sentences, before long node2vec [12] extended its work by introducing the biased random walk with hyper parameters. LINE [34] and SDNE [35] designed two objective functions that one preserves the first-order and the other preserves the second-order proximity. GraRep [6] captured different k-step information in a network by factorizing k-step similarity matrix and concatenating k-step embedding vectors together.

Recently, researchers have paid efforts to adapt existing network embedding approaches to dynamic networks. One distinctive task is the incremental update, i.e., to learn the new representation according to the original node representation during the evolvement of a network. To improve its computation efficiency, DANE [18], DHPE [44] and dyHNE [36] designed matrix factorization approaches with Matrix Perturbation Theory, therefore avoiding the re-train of the whole model. Compared with TPANE, these methods suffered from error accumulation [42]. NEU algorithm [40] provided approximation computation ways for high order proximity matrix. Skip-gram based models [20, 30] proposed a sampling trick for Deepwalk [25]. It indicated that only part of the generated random walks needs to be re-sampled when doing incremental updates on dynamic networks. Peng et al. rewrote the objective function in Skip-Gram with Negative Sampling (SGNS) model to accelerate the training of network embedding [24].

Another distinctive task in dynamic network embedding is to improve the effectiveness of the embedding by utilizing its temporal information. CTDNE [22] and dynnode2vec [20] have generalized Deepwalk [25] and node2vec [12] for dynamic networks by introducing temporal constraint to random walk, and have

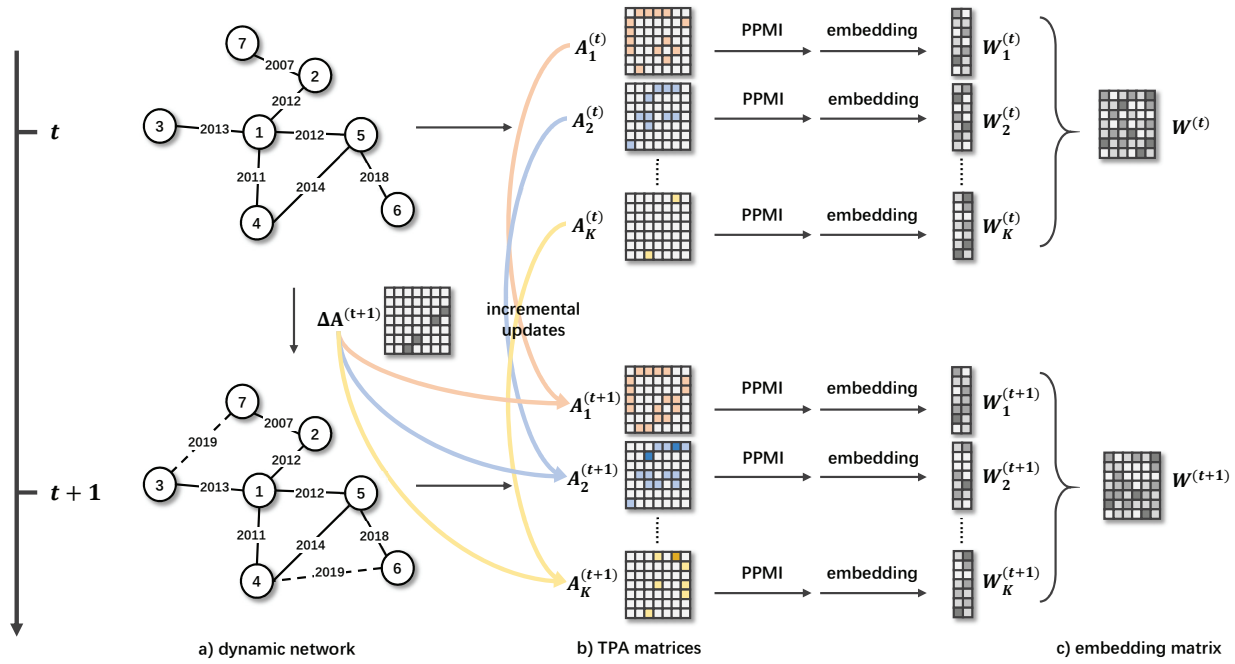

Figure 2: An illustration of the proposed dynamic network embedding model TPANE. At time step $t+1$, with two edges added into the dynamic network, TPANE updates TPA matrices $\{A_k^{(t+1)}\}$ according to the values of the previous TPA matrices $\{A_k^{(t)}\}$ and the increment of adjacency matrix $\Delta A^{(t+1)}$. Afterward, PPMI matrices are derived and the latest dynamic network embedding vectors are solved via Singular Value Decomposition.

verified its effectiveness by comparison experiments. HTNE [45] integrated the Hawkes process into network embedding vectors, assuming the influence of historical neighbors on current neighbors is decay over time. DynamicTraid [43] preserved both structural information and evolution patterns by discovering the triad unit in dynamic networks. Evolve2vec [2] proposed another temporal random walk generation algorithm that unfolded temporal edges into snapshots.

The stability of dynamic network embedding, also called adaptability [25], is also an essential issue. DynGEM [11] proposed the optimization objective of stability and minimize the differences between previous network embedding vectors and current network embedding vectors while incremental updating, which is meaningful to avoid the embedding space drifting. Du et al. presented a dynamic embedding framework [9] that is advantageous in enhancing network embedding efficiency and stability when doing the incremental update. It only adjusts the network embedding of the most affected nodes, which can be evaluated by the differences between the similarity matrix and its low-rank approximation.

3 NOTATIONS AND PROBLEM DEFINITIONS

In this section, the related notations and definitions of the dynamic network embedding problem are given.

Definition 1. *(Dynamic Network).* A dynamic network at time step t is denoted as $G^{(t)} = (V^{(t)}, E^{(t)})$, where $V^{(t)} = \{v_1, \cdots, v_N\}$ denotes the set of N nodes in $G^{(t)}$, $E^{(t)}$ the edges in $G^{(t)}$, and $A^{(t)} \in \mathbb{R}^{N\times N}$ the adjacency matrix at time step t. For ease of presentation, the number of nodes N is assumed to be constant over time. However, the proposed TPANE can naturally deal with node addition by extending the shape of TPA matrices and filling zero elements. $\Delta A^{(t+1)} = A^{(t+1)} - A^{(t)}$ denotes the change of the adjacency matrix from time step t to $t+1$, which enables the representation of the events of edge addition / deletion on the dynamic network. Deleting an edge weighted w from the network is equal to adding an edge weighed $-w$. Therefore, the dynamic network is defined as $G^{(t)} = \{\Delta A^{(1)}, \cdots, \Delta A^{(t)}\}$, where the initial adjacency matrix $A^{(0)}$ is assumed to be empty. To make theoretical explanation easier, we define $\mathcal{T} : E \to \mathbb{Z}^+$ as the function to map an edge to its arrival timestamp.

Definition 2. *(Dynamic Network Embedding).* Given a dynamic network $G^{(t)} = (V^{(t)}, E^{(t)})$, dynamic network embedding aims to learn a low dimensional vector $\vec{v} \in \mathbb{R}^d$ for every node $v \in V^{(t)}$ at time step t, where d denotes the dimension of embedding vectors. The embedding matrix of a dynamic network at time step t is therefore defined as $W^{(t)} \in \mathbb{R}^{N\times d}$, each row of which is the embedding vector of a node.

Definition 3. *(Temporal Path).* A k-step temporal path [22] from v_1 to v_k is defined as a path on a dynamic network

$$\text{TemporalPath}_k(v_1, v_k) = v_1 \xrightarrow{e_1} v_2 \xrightarrow{e_2} \cdots \xrightarrow{e_{k-1}} v_k, \qquad (1)$$

such that $\mathcal{T}(e_1) \le \mathcal{T}(e_2) \le \cdots \le \mathcal{T}(e_{k-1})$, where k is the length of temporal walk.

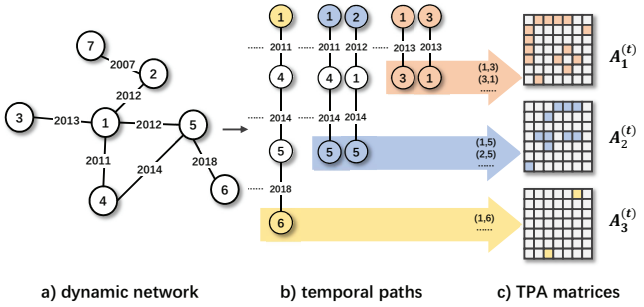

Figure 3: An illustration of Temporal Path Adjacency (TPA). Each temporal path from node u to v with length k contributes to an increment for $(A_k^{(t)})_{u,v}$.

4 TPANE MODEL

4.1 Temporal Path Adjacency Similarity Measure

To better introduce the Temporal Path Adjacency, we firstly introduce the similarity measure implicitly used in Deepwalk [27] in this sub-section. When the walk length $L \to \infty$, the probability for a transition from node u to node v in Deepwalk can be formulated as

$$p(v|u) = \Big(\sum_{k=1}^{w} P^k\Big)_{u,v} \tag{2}$$

where $P = D^{-1}A$ denotes the transition matrix, w denotes the window size [27]. It is clarified [6] the idea of k-step transition probability $p_k(v|u)$ as follow

$$p_k(v|u) = \big(P^k\big)_{u,v} \tag{3}$$

which corresponds to the probability for a random walk from u to v with k steps. Furthermore, it can be generalized to temporal random walks. Following the discussion of temporal random walk, the temporal neighborhood of node u is defined as

$$\Gamma_t(u) = \big\{v|(u,v) \in E^{(t)} \wedge \mathcal{T}(e=(u,v)) > t\big\}, \tag{4}$$

and the unbiased temporal neighborhood selection probability is defined as

$$\Pr(u) = \frac{1}{|\Gamma_t(u)|}. \tag{5}$$

Therefore, the k-step temporal transition probability is defined as the sum of each temporal random walk's probability contribution,

$$p_k(v|u) = \sum_{\text{TemporalWalk}_k(u,v)} \Pr(u)\cdots\Pr(v). \tag{6}$$

Although this similarity measure can gather both temporal and structural information from a dynamic network, it can hardly support incremental computation, which is explained in the example with Figure 1 in Section 1. To address the above-mentioned issue, the Temporal Path Adjacency (TPA) is proposed to evaluate the similarity between nodes on a dynamic network, which is inspired by the idea of Temporal Random Walk[22] and Temporal Katz Index[4].

In Temporal Path Adjacency, the number of different temporal walks between nodes is counted as similarity. Therefore, the contribution of each temporal walk is calculated independently, whose advantage will be shown in Section 4.2. To be specific, according to the definition of Temporal Path in Equation 1, the k-step TPA $A_k^{(t)}$ is defined as

$$(A_k^{(t)})_{u,v} = \Big|\big\{\text{TemporalPath}_k(u,v)\big\}\Big|, \tag{7}$$

and the illustration is given in Figure 3. It is noticeable that the adjacency matrix is a special case of the temporal adjacency matrix when $k = 1$. It can be used to derive the Temporal Katz Centrality [4] by summing up with different k-step TPA together and assigning exponentially decaying weight, which is applied in the PageRank Algorithm.

4.2 Updating Formula

Given the definition of k-step TPA matrix $A_k^{(t)}$ in Equation 7, it is important to compute efficiently it when a dynamic network is evolving. The updating formula and algorithms are provided in this section. When a new edge (w,v) arrives at time step $t+1$, it lengthens the existing temporal paths that end at w. Thus, the count of temporal paths from any node u to v with length $k+1$ can be derived by the following formula

$$(A_{k+1}^{(t+1)})_{u,v} = (A_{k+1}^{(t)})_{u,v} + (A_k^{(t)})_{u,w}. \tag{8}$$

Given $\{A_1^{(t)}, \cdots, A_K^{(t)}\}$ at time step t, if the network's adjacency matrix changes at time step $t+1$, denoted as $\Delta A^{(t+1)}$, then the TPA at next time step $\{A_1^{(t+1)}, \cdots, A_K^{(t+1)}\}$ can be updated by the following formulas

$$A_{k+1}^{(t+1)} = A_{k+1}^{(t)} + A_k^{(t)} \cdot \Delta A^{(t+1)}, \quad k \geq 1, \tag{9}$$

$$A_1^{(t+1)} = A_1^{(t)} + \Delta A^{(t+1)}. \tag{10}$$

Time Complexity. Since $A_k^{(t)}, \Delta A^{(t+1)}$ are sparse, the above algorithm can be sped up by sparse matrix multiplication techniques that transform the dense matrix multiplication into querying [41]. The time complexity of computation $A_k^{(t)} \cdot \Delta A^{(t+1)}$ is $O(N\delta M^{(t+1)})$ when ΔA is sparse [41], where $\delta M^{(t+1)}$ denotes the non-zero elements in $\Delta A^{(t+1)}$.

For online learning, i.e., to solve $A^{(t+1)}$ with $A^{(t)}$ already known, the time complexity of updating TPA is

$$O(KN\delta M^{(t+1)}) \tag{11}$$

which is linear with respect to the number of added edges. For offline learning, according to the T step iteration, the time complexity is

$$O\big(KN(\delta M^{(1)} + \cdots \delta M^{(T)})\big) = O(KNM) \tag{12}$$

where M denotes the number of edges in a dynamic network. It is noticeable that in each timestamp the computation of $A_k^{(t)}$ with different k can be processed in parallel with time complexity $O(NM)$. The offline algorithm for computing TPA is given in Algorithm 1. Notice that GraRep [6] is a special case of TPANE when the given network is static instead of dynamic ($T = 1$).

Algorithm 1: Temporal Path Adjacency

Input: current time step T, snapshots of dynamic network $\{\Delta A^{(1)}, \cdots, \Delta A^{(T)}\}$; maximum step of TPA K
Output: Temporal Path Adjacency matrices $\{A_1^{(T)}, \cdots, A_K^{(T)}\}$

for $k = 1$ *to* K **do**
  $A_k^{(0)} \leftarrow O$ (O is $N \times N$ zero matrix) ;
end
for $t = 1$ *to* T **do**
  $A_1^{(t)} \leftarrow A_1^{(t-1)} + \Delta A^{(t)}$;
  for $k = 2$ *to* K **do**
    $A_k^{(t)} \leftarrow A_k^{(t-1)} + A_{k-1}^{(t-1)} \cdot \Delta A^{(t)}$;
  end
end

Compared with temporal transition probability, the Temporal Path Adjacency is advantageous for the following reasons:

- It can be efficiently updated online. The latest similarity can be derived from the previous result with the computation complexity linear to the number of arriving edges.
- It yields precise similarity. By comparison, the estimation of temporal transition probability in CTDNE is based on random sampling, which meets the trade-off between the precision and the sample size.

4.3 Loss Function on Network

The Skip-Gram model with Negative Sampling (SGNS) model is widely used to learn the dynamic network embedding [12, 20, 22, 24, 25, 32], and proved to be equivalent to matrix factorization [27]. Following the discussion in [6], the k-step loss function for dynamic network embedding is as follows:

$$L_k = \sum_{v \in V} \sum_{u \in V} L_k(u, v). \tag{13}$$

For nodes u, v and their embedding vectors $\vec{u}, \vec{v}$, the k-step loss function is defined as

$$L_k(u, v) = q_k(v|u) \log \sigma(\vec{u} \cdot \vec{v}) + \lambda q_k(v) \log \sigma(-\vec{u} \cdot \vec{v}), \tag{14}$$

where $q_k(v|u)$ is the k-step relationship between u and v, $\sigma(x) = (1 + e^{-x})^{-1}$ is the Sigmoid function, and λ denotes the hyper-parameter for negative sampling. The former term in Equation 14 corresponds to positive sampling in skip-gram model, and the latter corresponds to negative sampling. In consideration of the imbalance of degrees between nodes, $q_k(v|u)$ uses the normalized TPA between u and v,

$$q_k(v|u) = \left(\bar{A}_k^{(t)}\right)_{u,v} = \frac{\left(A_k^{(t)}\right)_{u,v}}{\sum_{v'} \left(A_k^{(t)}\right)_{u,v'}}. \tag{15}$$

Assuming that the probability for each node to be selected is equal, the negative sampling distribution is defined as

$$q_k(v) = \frac{1}{N} \sum_{u'} q_k(v|u'). \tag{16}$$

By defining $e_{u,v} = \vec{u} \cdot \vec{v}$ and letting $\frac{\partial L_k}{\partial e_{u,v}} = 0$, we have

$$e_{u,v} = \vec{u} \cdot \vec{v} = \log\left(\left(\bar{A}_k^{(t)}\right)_{u,v}\right) - \log\left(\frac{\lambda}{N}\right), \tag{17}$$

which yields the similarity for embedding vectors. Then, the low-dimensional embedding problem can be transformed into a matrix factorization problem.

4.4 Optimization with Matrix Factorization

To better extract information via matrix factorization [17], the similarity matrix $(e_{u,v})_{N \times N}$ is cast into Positive PMI matrix $X \in \mathbb{R}^{N \times N}$ as follows,

$$\left(X_k\right)_{u,v} = \max(e_{u,v}, 0). \tag{18}$$

Singular Value Decomposition (SVD) can find a low-rank approximation for a given matrix with minimum squared error. It has shown great success in network embedding as a vital part of Matrix Factorization approaches [6, 19, 27]. For the given PPMI matrix X_k, the d-rank SVD factorizes it into

$$U_k \Sigma_k V_k^T \approx X_k, \tag{19}$$

where $U_k, V_k \in \mathbb{R}^{N \times d}$ and $\Sigma_k = \mathrm{diag}(\sigma_1, \cdots, \sigma_d)$. Therefore, the left and right factorized matrix $W_k, C_k \in \mathbb{R}^{N \times d}$ are

$$W_k = U_k \, \Sigma_k^{1/2}, \quad C_k = V_k \, \Sigma_k^{1/2}, \tag{20}$$

where W_k is the target k-step embedding matrix.

After calculating W_k for each k, the target embedding matrix is the concatenation of k-step embedding matrix where $1 \le k \le K$,

$$W = \left[W_1, \cdots, W_K\right] \in \mathbb{R}^{N \times Kd}. \tag{21}$$

and the target embedding vectors can be obtained from W,

$$W = \left[\vec{v_1}, \cdots, \vec{v_N}\right]^T. \tag{22}$$

Finally, the learning algorithm for the proposed network embedding method is given in Algorithm 2.

Algorithm 2: TPANE

Input: current time step t, dynamic network $G = \{\Delta A^{(1)} \cdots, \Delta A^{(t)}\}$, maximum step of Temporal Adjacency K, dimension of embedding vectors d, hyper-parameter for negative sampling λ
Output: Dynamic network embedding matrix $W = \left[\vec{v_1}, \cdots, \vec{v_N}\right]^T \in \mathbb{R}^{N \times d}$

$d' \leftarrow \lfloor d/K \rfloor$;
$\{A_1^{(t)}, \cdots A_K^{(t)}\} \leftarrow$ TemporalPathAdjacency(t, G, K) ;
for $k = 1$ *to* K **do**
  $\bar{A}_k^{(t)} \leftarrow$ Normalization$(A_k^{(t)})$;
  $X_k \leftarrow \max(\bar{A}^{(t)}, O)$;
  $U_k, \Sigma_k, V_k \leftarrow \mathrm{SVD}_{d'}(X_k)$;
  $W_k \leftarrow U_k \Sigma_k^{1/2} \in \mathbb{R}^{N \times d'}$
end
$W \leftarrow \left[W_1, \cdots, W_K\right] \in \mathbb{R}^{N \times d}$

Time Complexity. For online learning, the time complexity of TPANE is $O(N\delta M^{(t+1)} + M)$. For offline learning, the time

complexity of Algorithm 2 is $O(KNM + M) = O(KNM)$ according to Equation 12. Time complexity contributed by the SVD is not included since the time complexity estimation for low-rank SVD on a sparse matrix is vague. Given a dense $N \times N$ matrix, the time complexity of its full-rank SVD is $O(N^3)$. Fortunately, many works have been conducted to speed up SVD. Brand et al. proved that a rank-d thin SVD of a $N \times N$ dense matrix can be computed in $O(N^2 d)$ time for $d \leq \sqrt{N}$ [5]. FaLRR [37] also provided a fast algorithm to solve low-rank SVD whose time complexity per iteration is $O(Nd^2 + Nd)$. Experiments are conducted to verify the average time complexity of low-rank SVD on sparse matrix using ARPACK [14] as eigensolver in Section 5.3.

5 EXPERIMENTS

To validate the effectiveness and efficiency of the proposed TPANE method on dynamic networks, we conduct the comparison experiments on six read-world dynamic networks. In particular, the following questions are to be answered:

- *Effectiveness*: Is the proposed TPANE more effective on dynamic network embedding?
- *Efficiency*: Is the proposed TPANE faster than other baseline methods during incremental updating?
- *Parameter Sensitivity*: Does a different value of parameter K affect the performance of TPANE?

5.1 Experimental Setting

Datasets. Six real-world dynamic networks are collected from Network Data Repository [28] and SNAP [16]. The topological information of the listed networks is shown in Table 1, and the detailed descriptions are given below.

- **reptilia-tortoise-network-fi** [29] is an animal interaction network. The nodes represent animals and the edges represent interactions between animals.
- **soc-wiki-elec** [15] is a voting network of Wikipedia, where nodes indicate users and the edges indicate votes.
- **SFHH-conf-sensor** [7] is a human contact network where nodes represent humans and edges represent contacts.
- **ia-radoslaw-email** [21] is an internal email communication network between employees of a mid-sized manufacturing company. The nodes represent employees, and the edges represent individual emails between two users.
- **sx-mathoverflow-c2a** [23] is a temporal network of interactions on the stack exchange web site. Nodes represent users, and edges represent comments between users.
- **ia-contacts-dublin** [31] is a human contact network where nodes represent humans and edges between them represent contacts.

Baseline Methods. For comparison, five state-of-the-art network embedding methods are selected as the baseline methods.

- **Deepwalk** [25] is a static network embedding method that treats nodes as words, walks on a network as sentences, and learns network embedding via skip-gram.
- **LINE** [34] is a static network embedding method that preserves first-order and second-order proximity. In this experiment, both the first-order and second-order proximity are employed to learn representation.
- **DHPE** [44] is a dynamic network embedding method that adopts generalized SVD to preserve the high-order proximity. The static DHPE model, whose effectiveness is the upper bound of its dynamic model, is selected as the comparison baseline model in this experiment.
- **CTDNE** [22] is a dynamic network embedding method that can incorporate temporal information into embedding vectors via temporal random walk. It learns temporal dependency differently compared with TPANE.
- **GraRep** [6] is a static network embedding method that captures the different k-step relational information via matrix factorization. It adopts a similar network embedding framework compared with TPANE.

Table 1: Topological information of the dynamic network datasets.

	$\|V\|$	$\|E\|$	average degree
reptilia-tortoise-network-fi	787	1.7K	4
SFHH-conf-sensor	403	70.3K	348
ia-radoslaw-email	167	82.9K	992
soc-wiki-elec	7.1K	107K	30
sx-mathoverflow-c2a	13.8K	195K	14
ia-contacts-dublin	11K	415.9K	74

Parameter Setting. For Deepwalk and CTDNE, the number of walks $\mu = 100$, and the length of walk $l = 40$. For LINE, the number of negative samples is set to be 5 and all the embedding vectors are finally normalized by setting $\|\vec{v}\|_2 = 1$ following the origin parameters setting in [34]. For GraRep and TPANE, the maximal k-step size $K = 4$. For all the model, the embedding size $d = 128$.

Evaluation Metric. Area Under Curve (AUC) is used as the evaluation metric in our experiment, which represents the area under the Receiver Operating Characteristic (ROC) [26]. A higher AUC indicates better performance for the classification.

5.2 Evaluation of the Link Prediction

As a supervised classification task, link Prediction aims to predict whether there will be a link (a.k.a. edge) between two nodes based on the observed information of a network, which is widely used to evaluate the effectiveness of the network embedding models. 75% of the temporal edges on each datasets are fed into the network embedding model to learn the node embedding vectors. The rest of the 25% edges, together with the same number of generated negative edges (unconnected node pairs), are utilized for training and testing Logistic Regression (LR). For edge $e = (u, v)$, the edge embedding vector $\vec{e}$ is generated via the hadamard operator $\vec{e} = \vec{u} \odot \vec{v}$ or the mean operator $\vec{e} = (\vec{u} + \vec{v})/2$ [12]. The final results are averaged from 20 repeated experiments.

Results. The link prediction experiments were performed on five datasets, and the results were provided in Table 2 and Figure 4. Table 2 shows the performance of all the compared methods on the link prediction task, with 10% of edges used for training and 90% used for testing.

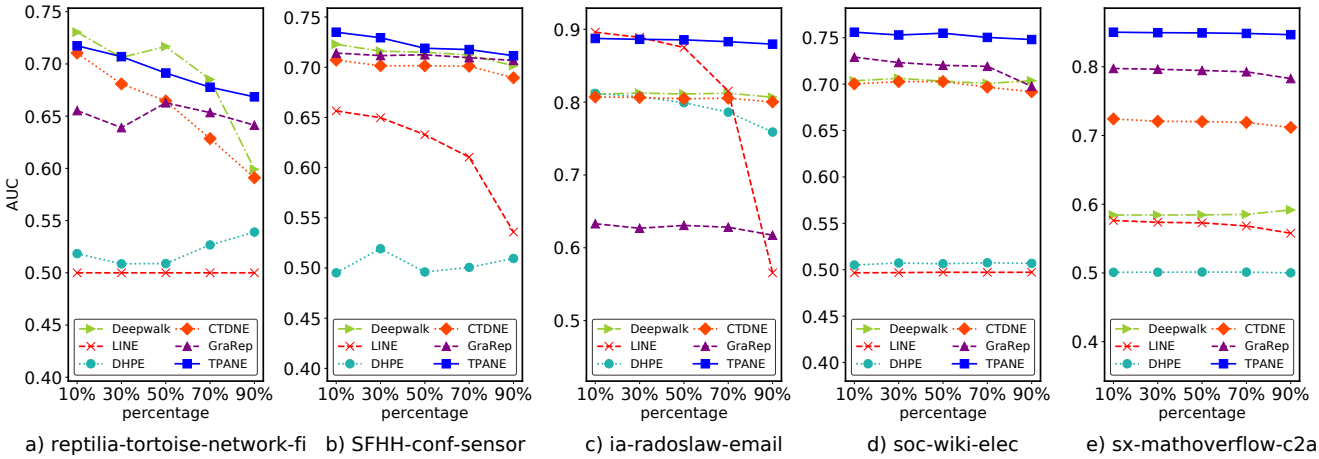

Figure 4: Link prediction results with hadamard operator. Percentage represents the proportion of edges for testing in classification.

Table 2: Link prediction results (10% training set, 90% testing set).

Evaluation Setting	Dataset	Method					
		Deepwalk	LINE	DHPE	CTDNE	GraRep	TPANE
Mean Operator	reptilia-tortoise-network-fi	0.6224	0.5024	0.5436	0.6154	0.6294	**0.6597**
	SFHH-conf-sensor	0.7000	0.6551	0.5467	**0.7089**	0.6723	0.6878
	ia-radoslaw-email	0.8304	0.8301	0.8416	0.8285	0.8160	**0.8485**
	soc-wiki-elec	0.7824	0.7052	0.6285	0.7845	0.8188	**0.8289**
	sx-mathoverflow-c2a	0.7358	0.8118	0.5187	0.8410	0.8757	**0.8865**
Hadamard Operator	reptilia-tortoise-network-fi	0.5991	0.5000	0.5390	0.5910	0.6414	**0.6685**
	SFHH-conf-sensor	0.7015	0.5358	0.5095	0.6895	0.7068	**0.7115**
	ia-radoslaw-email	0.8067	0.5656	0.7589	0.8002	0.6173	**0.8796**
	soc-wiki-elec	0.7037	0.4973	0.5069	0.6918	0.6977	**0.7481**
	sx-mathoverflow-c2a	0.5918	0.5578	0.5002	0.7117	0.7828	**0.8467**

Table 2 shows that nine of the ten best results are produced by TPANE. TPANE outperforms all state-of-the-art methods in all the cases with hadamard operator, and in most cases with mean operator. In some datasets like *soc-wiki-elec* and *sx-mathoverflow-c2a*, TPANE improves AUC by approximately 6% on AUC with hadamard operator compared with the other best methods. Figure 4 shows that the performance of TPANE is stably superior to other methods under different partition ratios between the training set and testing set in classification, especially on *SFHH-conf-sensor*, *soc-wiki-elec*, and *sx-mathoverflow-c2a*.

The proposed TPANE outperforms GraRep on all datasets regardless of the operator function, especially on *reptilia-tortoise-network-fi* with nearly 26% promotion on AUC with hadamard operator. Since the embedding framework of TPANE is the same as to GraRep except for the employment of TPA, the result demonstrates that the proposed TPA similarity measure can improve the quality of embedding vectors by learning the temporal dependency. Experiments with other ratios (65%, 85%) of observed edges for network embedding learning are conducted and similar conclusions can be drawn from the results (data not shown).

Above all, the results of link prediction experiments illustrate that TPANE is more effective than other state-of-the-art methods and can produce embedding vectors with higher quality.

5.3 Efficiency of low-rank SVD on sparse matrix

To demonstrate the efficiency of low-rank SVD on sparse matrix, we ran d-rank SVD on the adjacency matrices of synthetic networks with different scales.

Experimental Setting. We let the number of nodes $N = 1 \times 10^3, \cdots, 2 \times 10^5$, and average degree $M/N = 100$. The random networks were generated by Barabási–Albert preferential attachment algorithm [1]. ARPACK is employed as the eigensolver, which uses $N \cdot O(d) + O(d^2)$ storage [14].

Results. We generated the adjacency matrices with different sizes and applied d-rank SVD on them, and the time cost result was

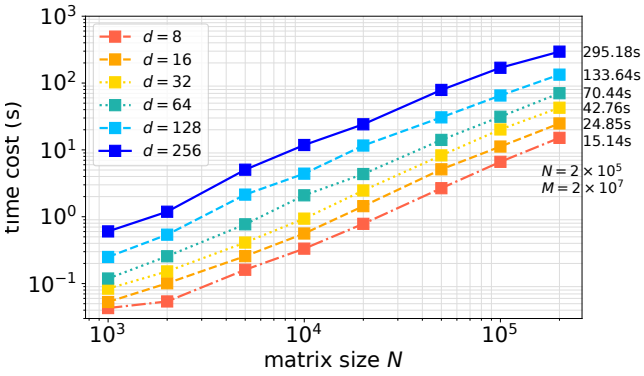

Figure 5: Efficiency of low-rank SVD on sparse matrix.

demonstrated in Figure 5, which was averaged from 20 repeated experiments. It shows that the average time complexity of SVD on sparse matrix is roughly $O(Nd)$ when M/N is small and $d << N$. The annotated points on Figure 5 show that when $N = 2 \times 10^5$, $M = 2 \times 10^7$ and $d = 128$, the time consumption of d-rank SVD is less than 3 minutes. Considering most of the network embedding models prefer $d \leq 200$, time consumption like this is acceptable.

Above all, if the average degree M/N is guaranteed to be small, matrix factorization approaches can also deal with networks with large scale in an efficient way comparing with random-walk-based approaches.

5.4 Efficiency on Incremental Updates

To evaluate the efficiency of our model, we conduct the time consumption analysis of network embedding models for incremental updates on dynamic networks with different scales.

Baseline Methods. Since CTDNE and GraRep do not naturally support the incremental updates, their online learning editions are provided, and explanations are given as follows:

- **CTDNE-unbiased-update** is an implementation of CTDNE [22] for incremental updates. It uses the Unbiased Update algorithm [30] to reduce unnecessary computation and improve efficiency.
- **GraRep-retrain** simply retrains the GraRep model [6] when incremental updating using GraRep methods.

Parameter Setting. For CTDNE-unbiased-update, the number of walks $\mu = 10$, which is less than the setting in link prediction task in Section 5.2, in order to make it faster, and the length of walk $l = 40$. For GraRep-retrain and TPANE, the maximal k-step size $K = 4$. For all the models, the embedding size $d = 128$.

Experimental Setting. On each dataset, the offline learning method is applied to learn the network embedding vectors. Then, some edges are generated as the dynamic network's new snapshot, and the online learning method is applied to learn the new network embedding vectors. The time consumption of the online learning is recorded.

Results. The comparison experiments are conducted on six real-world dynamic networks, and the results are given in Figure 6. As shown in Figure 6, TPANE is faster than both the two baseline methods on datasets with various scales. For CTDNE-unbiased-update, its time consumption is mainly contributed by the generation of the updated walks and the iteration of gradient descent. Therefore, it is about 100 times slower than the other matrix factorization methods. It is amazing that TPANE is even faster than GraRep-retrain, since the GraRep method does really little computation and simply employs SVD on normalized different order adjacency matrices. Figure 6 also shows that the time consumption of TPANE is about 40% less than GraRep-retrain on large dynamic networks like *sx-mathoverflow-c2a* and *ia-contacts-dublin*, since TPANE needs to do less matrix multiplication while doing incremental updates. The experiment result illustrates that TPANE is efficient for incremental update, which is vital for online learning of dynamic network embedding.

5.5 Parameter Sensitivity

To answer how the parameters in TPANE affect its effectiveness and efficiency, we apply different values of parameter $K = 2, 4, 6, \cdots, 16$ on TPANE, and perform the link prediction experiment as well as the time consumption experiment. Results are shown in Figure 7 and Figure 8.

Figure 7 shows the performance of TPANE on link prediction tasks under different parameter settings. It illustrates that the perturbation amplitude of AUC is small (less than 5% if K is greater than 4), therefore showing that TPANE is insensitive to the parameter K. The effectiveness of TPANE falls when K increases can be explained by the fact that the high step TPA is not as important as the low step TPA. When k is getting larger and larger, the dimensions of embedding vectors assigned to each k-step TPA decreases, and therefore the learning of low step TPA is disturbed. For *SFHH-conf-sensor*, the optimal parameter setting is $K = 4$, as well as $K = 2$ for *ia-radoslaw-email* and $K = 4$ for *soc-wiki-elec*.

Figure 8 shows the time cost of TPANE when doing incremental updates under different parameter settings when $\delta M = 250$. It shows that the time cost of TPANE for incremental updates is roughly linear to the parameter K. According these two experiment results, to optimize the effectiveness of TPANE and keep efficient, K should be selected as a small number and $K = 4$ is a good default option for most of the cases.

6 CONCLUSIONS

In this article, an efficient and effective embedding model for dynamic networks named **TPANE** is proposed. It preserves both the temporal dependency and the topological information between nodes, and updates the embedding vectors of a dynamic network via matrix factorization. A new similarity measure TPA is also proposed, which is capable of incremental computation and learning temporal information. Therefore, it is practical in dynamic network embedding and can be used to construct new similarity measures. Experiments on six real-world dynamic networks demonstrate that TPANE is effective and outperforms state-of-the-art network embedding methods. Time consumption analysis shows that TPANE is faster than baseline methods while doing incremental updates. Future work includes generalizing TPANE into various networks like heterogeneous network and constructing new similarity measures based on TPA.

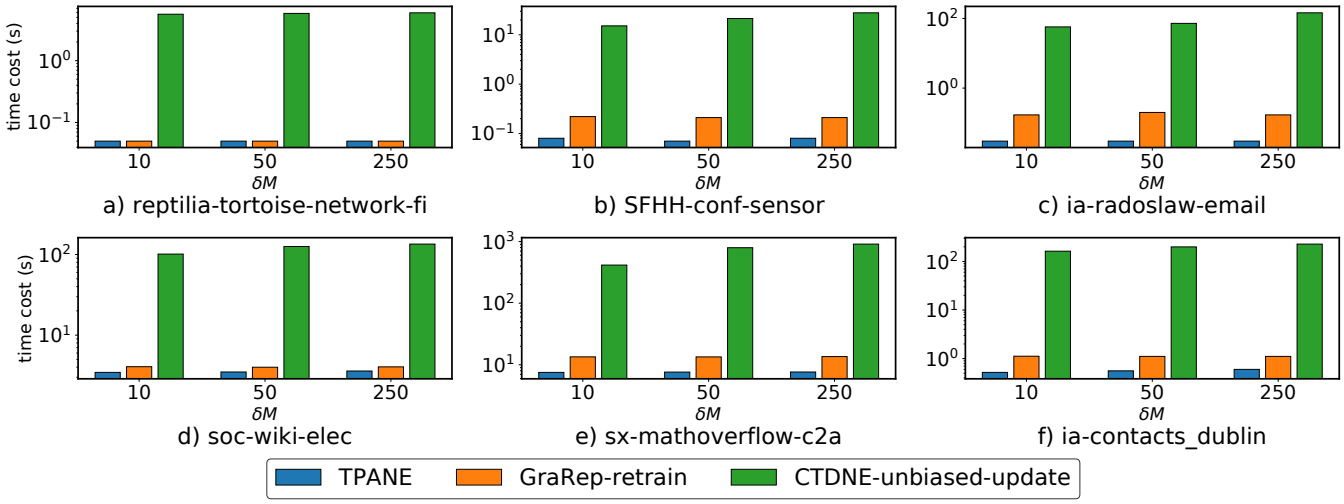

Figure 6: Time consumption for incremental updates with different update rate δM.

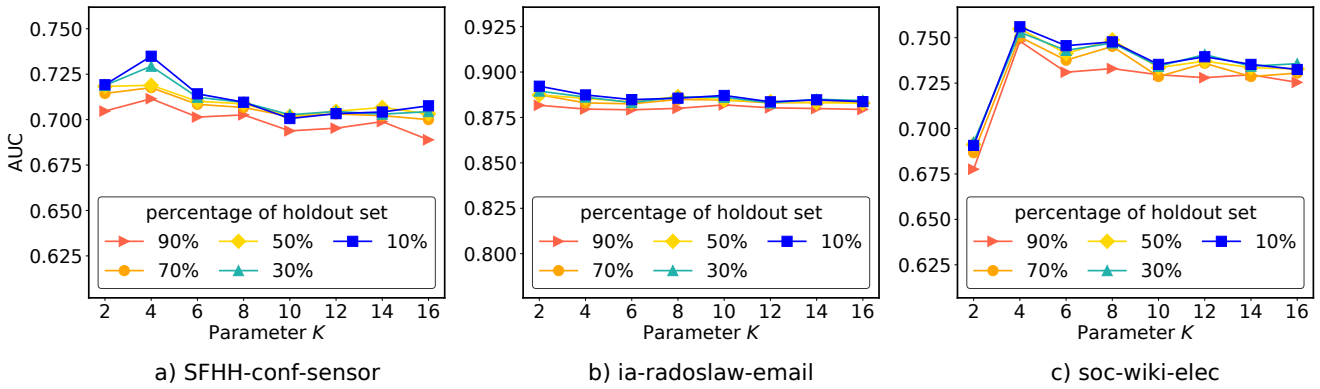

Figure 7: Performance on link prediction with different maximum k-step size K.

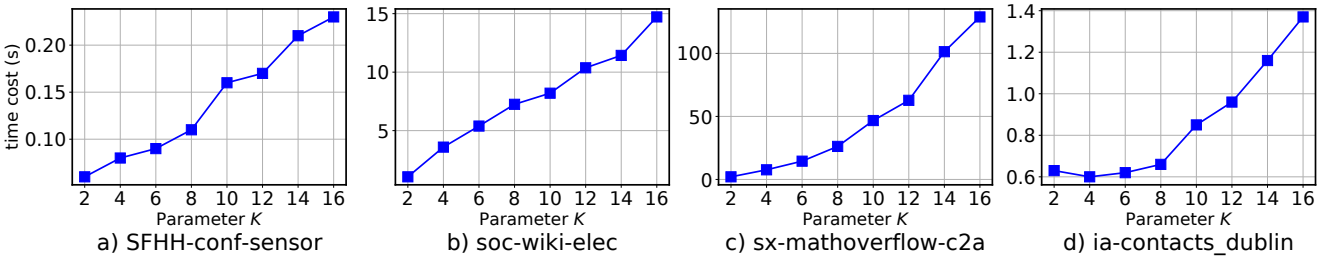

Figure 8: Time cost for incremental updates with different maximum k-step size K ($\delta M = 250$).

ACKNOWLEDGMENTS

This work was supported by the National Key Research and Development Program of China under Grant No.2019YFB2102200 and by the National Natural Science Foundation of China under Grant No.61202262.

REFERENCES

[1] Albert-László Barabási and Réka Albert. 1999. Emergence of scaling in random networks. *science* 286, 5439 (1999), 509–512.

[2] Nikolaos Bastas, Theodoros Semertzidis, Apostolos Axenopoulos, and Petros Daras. 2019. evolve2vec: Learning network representations using temporal unfolding. In *International Conference on Multimedia Modeling*. Springer, 447–458.

[3] Mikhail Belkin and Partha Niyogi. 2003. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation* 15, 6 (2003), 1373–1396.

[4] Ferenc Béres, Róbert Pálovics, Anna Oláh, and András A Benczúr. 2018. Temporal walk based centrality metric for graph streams. *Applied network science* 3, 1 (2018), 1–26.

[5] Matthew Brand. 2006. Fast low-rank modifications of the thin singular value decomposition. *Linear algebra and its applications* 415, 1 (2006), 20–30.

[6] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management*. 891–900.

[7] Ciro Cattuto, Wouter Van den Broeck, Alain Barrat, Vittoria Colizza, Jean-François Pinton, and Alessandro Vespignani. 2010. Dynamics of person-to-person interactions from distributed RFID sensor networks. *PloS one* 5, 7 (2010), e11596.

[8] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. 2018. A survey on network embedding. *IEEE transactions on knowledge and data engineering* 31, 5 (2018), 833–852.

[9] Lun Du, Yun Wang, Guojie Song, Zhicong Lu, and Junshan Wang. 2018. Dynamic Network Embedding: An Extended Approach for Skip-gram based Network Embedding.. In *IJCAI*, Vol. 2018. 2086–2092.

[10] Palash Goyal and Emilio Ferrara. 2018. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems* 151 (2018), 78–94.

[11] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. 2018. Dyngem: Deep embedding method for dynamic graphs. *arXiv preprint arXiv:1805.11273* (2018).

[12] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 855–864.

[13] Mingliang Hou, Jing Ren, Da Zhang, Xiangjie Kong, Dongyu Zhang, and Feng Xia. 2020. Network embedding: Taxonomies, frameworks and applications. *Computer Science Review* 38 (2020), 100296.

[14] Richard B Lehoucq, Danny C Sorensen, and Chao Yang. 1998. *ARPACK users' guide: solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods*. SIAM.

[15] Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. 2010. Signed networks in social media. In *Proceedings of the SIGCHI conference on human factors in computing systems*. 1361–1370.

[16] Jure Leskovec and Rok Sosič. 2016. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)* 8, 1 (2016), 1–20.

[17] Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. *Advances in neural information processing systems* 27 (2014).

[18] Jundong Li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. 2017. Attributed network embedding for learning in a dynamic environment. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 387–396.

[19] Xin Liu, Tsuyoshi Murata, Kyoung-Sook Kim, Chatchawan Kotarasu, and Chenyi Zhuang. 2019. A general view for network embedding as matrix factorization. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. 375–383.

[20] Sedigheh Mahdavi, Shima Khoshraftar, and Aijun An. 2018. dynnode2vec: Scalable dynamic network embedding. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 3762–3765.

[21] Radosław Michalski, Sebastian Palus, and Przemysław Kazienko. 2011. Matching organizational structure and social network extracted from email communication. In *International conference on business information systems*. Springer, 197–206.

[22] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. 2018. Continuous-time dynamic network embeddings. In *Companion Proceedings of the The Web Conference 2018*. 969–976.

[23] Ashwin Paranjape, Austin R Benson, and Jure Leskovec. 2017. Motifs in temporal networks. In *Proceedings of the tenth ACM international conference on web search and data mining*. 601–610.

[24] Hao Peng, Jianxin Li, Hao Yan, Qiran Gong, Senzhang Wang, Lin Liu, Lihong Wang, and Xiang Ren. 2020. Dynamic network embedding via incremental skip-gram with negative sampling. *Science China Information Sciences* 63, 10 (2020), 1–19.

[25] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 701–710.

[26] F Provost and T Fawcett. 1997. Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions In: Proc of the 3rd International Conference on Knowledge Discovery and Data Mining. (1997).

[27] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. 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*. 459–467.

[28] Ryan Rossi and Nesreen Ahmed. 2015. The network data repository with interactive graph analytics and visualization. In *Twenty-ninth AAAI conference on artificial intelligence*.

[29] Pratha Sah, Kenneth E Nussear, Todd C Esque, Christina M Aiello, Peter J Hudson, and Shweta Bansal. 2016. Inferring social structure and its drivers from refuge use in the desert tortoise, a relatively solitary species. *Behavioral Ecology and Sociobiology* 70, 8 (2016), 1277–1289.

[30] Hooman Peiro Sajjad, Andrew Docherty, and Yuriy Tyshetskiy. 2019. Efficient representation learning using random walks for dynamic graphs. *arXiv preprint arXiv:1901.01346* (2019).

[31] SocioPatterns. 2011. Infectious contact networks. http://www.sociopatterns.org/datasets/

[32] Yifan Song, Darong Lai, Zhihong Chong, and Zeyuan Pan. 2021. Dynamic Network Embedding by Time-Relaxed Temporal Random Walk. In *International Conference on Neural Information Processing*. Springer, 426–437.

[33] Gilbert W Stewart. 1990. Matrix perturbation theory. (1990).

[34] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*. 1067–1077.

[35] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 1225–1234.

[36] Xiao Wang, Yuanfu Lu, Chuan Shi, Ruijia Wang, Peng Cui, and Shuai Mou. 2020. Dynamic heterogeneous information network embedding with meta-path based proximity. *IEEE Transactions on Knowledge and Data Engineering* (2020).

[37] Shijie Xiao, Wen Li, Dong Xu, and Dacheng Tao. 2015. FaLRR: A fast low rank representation solver. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4612–4620.

[38] Yu Xie, Chunyi Li, Bin Yu, Chen Zhang, and Zhouhua Tang. 2020. A survey on dynamic network embedding. *arXiv preprint arXiv:2006.08093* (2020).

[39] Guotong Xue, Ming Zhong, Jianxin Li, Jia Chen, Chengshuai Zhai, and Ruochen Kong. 2022. Dynamic network embedding survey. *Neurocomputing* 472 (2022), 212–223.

[40] Cheng Yang, Maosong Sun, Zhiyuan Liu, and Cunchao Tu. 2017. Fast network embedding enhancement via high order proximity approximation.. In *IJCAI*, Vol. 17. 3894–3900.

[41] Raphael Yuster and Uri Zwick. 2005. Fast sparse matrix multiplication. *ACM Transactions On Algorithms (TALG)* 1, 1 (2005), 2–13.

[42] Ziwei Zhang, Peng Cui, Jian Pei, Xiao Wang, and Wenwu Zhu. 2018. Timers: Error-bounded svd restart on dynamic networks. In *Thirty-second AAAI conference on artificial intelligence*.

[43] Lekui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. 2018. Dynamic network embedding by modeling triadic closure process. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

[44] Dingyuan Zhu, Peng Cui, Ziwei Zhang, Jian Pei, and Wenwu Zhu. 2018. High-order proximity preserved embedding for dynamic networks. *IEEE Transactions on Knowledge and Data Engineering* 30, 11 (2018), 2134–2144.

[45] Yuan Zuo, Guannan Liu, Hao Lin, Jia Guo, Xiaoqian Hu, and Junjie Wu. 2018. Embedding temporal network via neighborhood formation. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2857–2866.