Accelerating Dynamic Network Embedding with Billions of Parameter Updates to Milliseconds

Haoran Deng
Zhejiang University
denghaoran@zju.edu.cn

Yang Yang*
Zhejiang University
yangya@zju.edu.cn

Jiahe Li
Zhejiang University
jiahe.20@intl.zju.edu.cn

Haoyang Cai
Carnegie Mellon University
hcai2@andrew.cmu.edu

Shiliang Pu
Hikvision Research Institute
pushiliang.hri@hikvision.com

Weihao Jiang
Hikvision Research Institute
jiangweihao5@hikvision.com

ABSTRACT

Network embedding, a graph representation learning method illustrating network topology by mapping nodes into lower-dimension vectors, is challenging to accommodate the ever-changing dynamic graphs in practice. Existing research is mainly based on node-by-node embedding modifications, which falls into the dilemma of efficient calculation and accuracy. Observing that the embedding dimensions are usually much smaller than the number of nodes, we break this dilemma with a novel dynamic network embedding paradigm that rotates and scales the axes of embedding space instead of a node-by-node update. Specifically, we propose the Dynamic Adjacency Matrix Factorization (DAMF[1]) algorithm, which achieves an efficient and accurate dynamic network embedding by rotating and scaling the coordinate system where the network embedding resides with no more than the number of edge modifications changes of node embeddings. Moreover, a dynamic Personalized PageRank is applied to the obtained network embeddings to enhance node embeddings and capture higher-order neighbor information dynamically. Experiments of node classification, link prediction, and graph reconstruction on different-sized dynamic graphs suggest that DAMF advances dynamic network embedding. Further, we unprecedentedly expand dynamic network embedding experiments to billion-edge graphs, where DAMF updates billion-level parameters in less than 10ms.

CCS CONCEPTS

• **Mathematics of computing** → **Graph algorithms**; • **Theory of computation** → **Dynamic graph algorithms**.

KEYWORDS

Dynamic Graphs; Network Embedding; Matrix Factorization; Graph Representation Learning

*Corresponding author.
[1]The code is available at https://github.com/zjunet/DAMF

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
KDD '23, August 6–10, 2023, Long Beach, CA, USA.
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0103-0/23/08…$15.00
https://doi.org/10.1145/3580305.3599250

ACM Reference Format:
Haoran Deng, Yang Yang, Jiahe Li, Haoyang Cai, Shiliang Pu, and Weihao Jiang. 2023. Accelerating Dynamic Network Embedding with Billions of Parameter Updates to Milliseconds. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3580305.3599250

1 INTRODUCTION

Network embedding is an advanced graph representation learning method that maps each node in a graph to a vector in a low-dimensional space while preserving the graph's topological information [2]. It is a versatile approach with successful practical applications in a wide range of fields, such as recommendation systems and bioinformatics [6, 7, 24, 32, 34, 41].

In practice, however, changes in graph structure are frequent and inevitable. For instance, in a social network, the nodes representing the new members are added in connection to the original network over time, forming new edges. On such dynamic graphs, the network embeddings should be updated with the transformation of the graphs to empower the model to capture crucial insights, such as which groups are more active or which members are more likely to be influencers. Moreover, the temporal evolution contributes to the portrait of the new members. Unfortunately, the frequent evolving patterns and the extensive network scale require enormous time and space to retrain the network embedding to model the dynamic network effectively.

Recent research has encountered a dilemma in efficiency and effectiveness. That is, precisely modifying the network embeddings leads to excessive inefficiencies. Some methods [5, 50, 54] choose to adjust the embedding of nearly every node (also known as global updates), resulting in high time complexity. On the other hand, some works [8, 15, 19, 21] make trade-offs by updating only the more affected nodes in the graph (also known as local updates) to ensure good efficiency, but constantly bringing errors and, consequently, leading to performance degradation. Moreover, existing applications like short video instant recommendations require high instantaneity, which indicates it is inappropriate to use delayed retraining methods, like gathering multiple modifications for a single update.

We break the dilemma by rotating and scaling the coordinate axes of the embedding space instead of updating individually from the nodes' perspective. By calculating the space projection matrices, the newly added edge is captured while retaining the semantics of the embedding from the previous step. By means of an embedding

space projection matrix and a small number of modifications of the node embedding, each node vector embedding in the graph will be relocated at the updated position in the embedding space. This approach is efficient since the number of coordinate axes (i.e., the dimension of the embedding) is significantly fewer than the number of nodes while retaining a high accuracy level compared to local updates.

In light of the preceding ideas, we propose the Dynamic Adjacency Matrix Factorization (DAMF) algorithm. The corresponding space projection matrix is solved based on *Zha-Simon's t-SVD update formula* [46], with additional modification at most Δm nodes' embedding, where Δm is the number of edge changes in the graph. Further, inspired by the great success of the application of Personalized PageRank (PPR) to static network embeddings [38, 42, 43] and graph neural networks [9], we use a dynamic PPR to enhance the network embedding in order to capture high-order neighbors' information.

With the above design and ideas, DAMF also provides theoretical guarantees of effectiveness and efficiency. For effectiveness specifically, let $\widetilde{\mathbf{A}}$ be the low-rank approximation of the adjacency matrix represented by current network embedding, the unenhanced DAMF achieves the matrix factorization of the updated $\widetilde{\mathbf{A}}$ with node or edge change (Theorem 1 & 2). Moreover, the dynamic embedding enhancement converges into accurate PPR propagation. For efficiency, the time complexity of the DAMF algorithm is proved to be $O(\Delta m)$ when hyperparameters are considered constants.

In addition, we are the first to extend dynamic network embedding experiments to billion-edge graphs, which is a breakthrough in the scale of massive graphs. We conduct our experiment on Twitter, a real-world graph dataset with 41 million nodes and **1.4 billion** edges, and map each node to a 128-dimensional vector with the number of learnable parameters more than 10 times that the BERT-Large's [16]. We add nodes to the graph and updated the network embeddings individually, starting from 1000 nodes. The total updating time of DAMF is 110 hours, illustrating that DAMF achieves billion-level parameter updates in less than 10ms. In addition, we have conducted experiments on node classification, link prediction, and graph reconstruction on graphs of different sizes. The experimental results show that DAMF reaches a new state of the art in dynamic network embedding.

To summarize, we make the following contributions:

- We present the DAMF algorithm, a novel dynamic network embedding method based on embedding space projections, and augment the embedding with dynamic PPR to capture higher-order neighbourhood information.
- We theoretically illustrate the efficiency and effectiveness of DAMF.
- To the best of our knowledge, we are the first to extend dynamic network embedding to a dataset with billions of edges. Experimental results show that our proposed methods use only an average of $10ms$ to complete global parameter updates. We also conduct experiments on five other actual network data, and the results show that the proposed methods reach a new state of the art in dynamic network embedding.

Table 1: Notations

Notation	Description
$\mathcal{G}(\mathcal{V}, \mathcal{E})$	the graph with node set $\mathcal{V}$ and edge set $\mathcal{E}$
n, m	the number of nodes and the number of edges
Δm	the number of edges change in graph
$deg(u)$	the degree (in-degree or out-degree) of node u
$\mathbf{A}$, $\mathbf{D}$	the adjacency matrix and degree matrix
$\mathbf{B}$, $\mathbf{C}$	the low-rank representation of the updated matrix.
$\mathbf{X}$, $\mathbf{Y}$	the context embedding and content embedding
$\mathbf{Z}$	the enhanced context embedding
$\mathbf{F}$, $\mathbf{G}$	the space projection matrix for $\mathbf{X}$ and $\mathbf{Y}$
$\Delta\mathbf{X}$, $\Delta\mathbf{Y}$	the node vectors that are directly modified
d	the dimension of the embedding space
α	the damping factor in *Personalized PageRank*
ϵ	the error tolerance in *Personalized PageRank*

2 PRELIMINARIES

Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ denotes the node set of n nodes and $\mathcal{E}$ denotes the edge set of m edges. *Dynamic graph scenario* is a sequential update events $\{Event_1, ..., Event_T\}$ with an initial graph $\mathcal{G}_0$. Each event is one of either node change or edge change. Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ be the adjacency matrix of $\mathcal{G}$, and $\mathbf{D} = diag\{deg[1], deg[2], ..., deg[n]\}$ be the diagonal degree matrix where $\deg[i] = \sum_j \mathbf{A}[i, j]$ is the out-degree of i-th node. Network embedding aims at mapping each node in graph $\mathcal{G}$ to one or two low-dimensional vectors, which capture each node's structural information in the graph. In this paper, for a given dimension $d \ll n$, the i-th node in graph $\mathcal{G}$ is mapped to two vectors $\mathbf{X}[i], \mathbf{Y}[i] \in \mathbb{R}^{\frac{d}{2}}$ with equal dimension, which capture the structural information. Dynamic Network Embedding updates the result of the embedding for the t-th event, and the snapshot of the updated embedding is noted as $\mathbf{X}_t, \mathbf{Y}_t$.

In this paper, matrices are denoted in bold uppercase letters. Let $\mathbf{M}$ be arbitrary matrix, $\mathbf{M}[i]$ is the i-th row vector of $\mathbf{M}$, $\mathbf{M}[:, j]$ is the j-th column vector of $\mathbf{M}$, and $\mathbf{M}[i, j]$ is the element on the i-th row j-th column of $\mathbf{M}$. In addition, we use $\mathbf{M}[: l]$ to denote the submatrix consisting of the first l rows of $\mathbf{M}$, and use $\mathbf{M}[l :]$ to denote the submatrix consisting of the remaining part of $\mathbf{M}$.

3 RELATED WORK

3.1 Network Embedding

Network embedding aims to reflect the structural information of nodes in a graph by mapping each node into a low-dimensional vector [14]. Compared to another popular graph learning method, Graph Neural Networks (GNNs), it has no requirement for any features or attributes of nodes.

Studies on network embedding can simply be classified into gradient-based methods and matrix factorization-based methods. Gradient-based methods like DeepWalk [28], node2vec [12], and LINE [35] learn a skip-gram with negative sampling from random-walk on the graph, while GraphGAN [40], GA [1], DNGR [4] use deep learning method to learn network embedding. For matrix factorization-based methods, AROPE [51], NetMF [31], NetSMF [30],

and LightNE [29] construct a matrix that reflects the properties of the graph and factorizes it to obtain the network embedding, while ProNE [48] and NRP [42] obtain network embedding by propagates the embedding on the graph after the factorization.

Among the network embedding methods, those who use PageRank [26], such as VERSE [38], STRAP [44], and NRP [42], have achieved an ideal result since they better capture the information of high-order neighbors.

3.2 Dynamic Network Embedding

Numerous graphs in the industry are dynamic with frequent node or edge modifications, which leads to the emergence of network embedding methods on dynamic graphs. Methods for dynamic graph embedding can be categorized as node-selection-based, matrix-factorization-based, and others.

Node-selection-based methods. Node-selection-based methods choose to update only a limited number of embeddings node-by-node, resulting in poor performance. DNE [8] updates the embedding of nodes by adapting the skip-gram to the dynamic graph; Dynnode2vec [21] fine-tunes the previous result with the newly sampled data; GloDyNE [15] improves the node selection strategy to guarantee a global topological relationship; LocalAction [19] achieves an efficient and dynamic network but with poor performance by randomly selecting a small number of neighbors around the updated nodes and modifying their embedding.

Matrix-Factorization-based methods. Matrix-factorization-based methods prefer global updates that adjust the embedding of almost every node, which leads to high time complexity. TRIP [5] efficiently updates the top eigen-pairs of the graph, but leads to a significant accumulation of errors; TIMERS [50] uses recalculation to mitigate the errors caused by TRIP; RandNE [49] uses a random projection method to update the factorization of the adjacency matrix; DHEP [54] modifies the most affected eigenvectors using the matrix perturbation theory. Unfortunately, regardless of the amount of changes, the time complexity of these methods except RandNE for a graph with n nodes is at least $O(n)$.

Other methods. DynGEM [11] and NetWalk [45] use an auto-encoder to continuously train the model with parameters inherited from the last time step with a regularization term. DepthLGP [20] considers adding nodes to the dynamic graph as if they were out-of-sample and interpolating their embedding. However, the above methods are difficult to adapt to frequently changed graphs or cold start scenarios.

As a distinction, the dynamic network embedding in this paper tries to adapt embedding updates to dynamically changing graphs instead of mining the graph for temporal information(e.g., DynamicTraid [53], CTDNE [25], DTINE [10], tNE[33]), and the nodes have no attributes or features(e.g., EvolveGCN [27], DANE [18]).

4 METHODOLOGY

In this section, we develop the Dynamic Adjacency Matrix Factorization (DAMF) algorithm. Figure 1 shows an overview of the proposed DAMF algorithm. In what follows, Section 4.1 introduces the concept of dynamic network embeddings based on space projections. Section 4.2 and Section 4.3 demonstrate how to modify the space projection matrix and the node embedding for node and edge changes, respectively. Section 4.4 presents an enhanced approach for dynamic network embedding using dynamic Personalized PageRank. Section 4.5 gives the detailed steps of the DAMF. Section 4.6 analyzes the time and space complexity of DAMF.

4.1 Dynamic Embedding via Space Projection

The truncated singular value decomposition(t-SVD) of the adjacency matrix of a graph provides an ideal network embedding. For a graph $\mathcal{G}$ with adjacency matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, this method provides a network embedding $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{n\times d}$ with dimension d by

$$\mathbf{U}, \Sigma, \mathbf{V} \leftarrow \texttt{t-SVD}(\mathbf{A}, d), \quad \mathbf{X} \leftarrow \mathbf{U}\sqrt{\Sigma}, \quad \mathbf{Y} \leftarrow \mathbf{V}\sqrt{\Sigma}, \tag{1}$$

In the scenario of a dynamic graph, the adjacency matrix changes over time, causing its t-SVD to change as well. However, the instantaneous recalculation of t-SVD is time-consuming, especially for large-scale adjacency matrices.

To cope with this problem, we propose a novel paradigm for updating network embedding by rotating and scaling (i.e., space projection) the space coordinate axes of embedding, with only a small number of nodes to update additionally. Specifically, we take the network embedding $\mathbf{X}, \mathbf{Y}$ at time $t-1$, rotate and scale its coordinate axis by the space projection matrices $\mathbf{F}, \mathbf{G} \in \mathbb{R}^{d\times d}$, then add ΔX and ΔY respectively to get the updated network embeddings $\mathbf{X}_t$ and $\mathbf{Y}_t$ at time t in the new coordinate system by

$$\mathbf{X}_t \leftarrow \mathbf{X}_{t-1}\cdot\mathbf{F} + \Delta\mathbf{X}_t, \quad \mathbf{Y}_t \leftarrow \mathbf{Y}_{t-1}\cdot\mathbf{G} + \Delta\mathbf{Y}_t \tag{2}$$

where the **number of non-zero rows** in $\Delta\mathbf{X}_t$ and $\Delta\mathbf{Y}_t$ is the number of nodes that need to be modified for embedding.

Nevertheless, the complexity of computing $\mathbf{X}_{t-1}\cdot\mathbf{F}$ and $\mathbf{Y}_{t-1}\cdot\mathbf{G}$ is very high. To address this issue, we map all graph modifications to a base space that stores network embeddings at any given timestamp. By matrix multiplication, successive space projections can be merged. To improve efficiency, we apply a "lazy" query optimization that keeps the accumulated modifications until a query of new embedding appears.

Specifically, $\mathbf{X}_0$ and $\mathbf{Y}_0$ are the initialized network embeddings of $\mathbf{X_b}, \mathbf{Y_b} \in \mathbb{R}^{n\times d}$ at timestamp 0, while the space projection matrix $\mathbf{P_X}, \mathbf{P_Y} \in \mathbb{R}^{d\times d}$ are initialized with the identity matrix. The operation in Eq. (2) can be transformed into

$$\begin{aligned} \mathbf{P}_{\mathbf{X},t} &\leftarrow \mathbf{P}_{\mathbf{X},t-1}\cdot\mathbf{F}, \quad \mathbf{X}_{b,t} \leftarrow \mathbf{X}_{b,t-1} + \Delta\mathbf{X}_t\cdot\mathbf{P}^{-1}_{\mathbf{X},t} \\ \mathbf{P}_{\mathbf{Y},t} &\leftarrow \mathbf{P}_{\mathbf{Y},t-1}\cdot\mathbf{G}, \quad \mathbf{Y}_{b,t} \leftarrow \mathbf{Y}_{b,t-1} + \Delta\mathbf{Y}_t\cdot\mathbf{P}^{-1}_{\mathbf{Y},t} \end{aligned} \tag{3}$$

Then, the space projection matrix and node embedding will be modified for each change to the graph.

4.2 Node Change

Adding nodes to a graph is equivalent to adding an equal number of rows and columns to its adjacency matrix. Without losing generality, we consider the newly added node is in the last column and row of adjacency matrix, and we treat the column and row addition as two individual steps, that is,

$$\mathbf{A}' \leftarrow [\mathbf{A}_{t-1}, \mathbf{B}_1], \quad \mathbf{A}_t \leftarrow [\mathbf{A}'^{\top}, \mathbf{B}_2]^{\top} \tag{4}$$

Because adding a row and adding a column are symmetrical, we describe our method in terms of adding a column to reduce redundant statements, and we use $\mathbf{B}$ to denote $\mathbf{B}_1$ and $\mathbf{B}_2$ above.

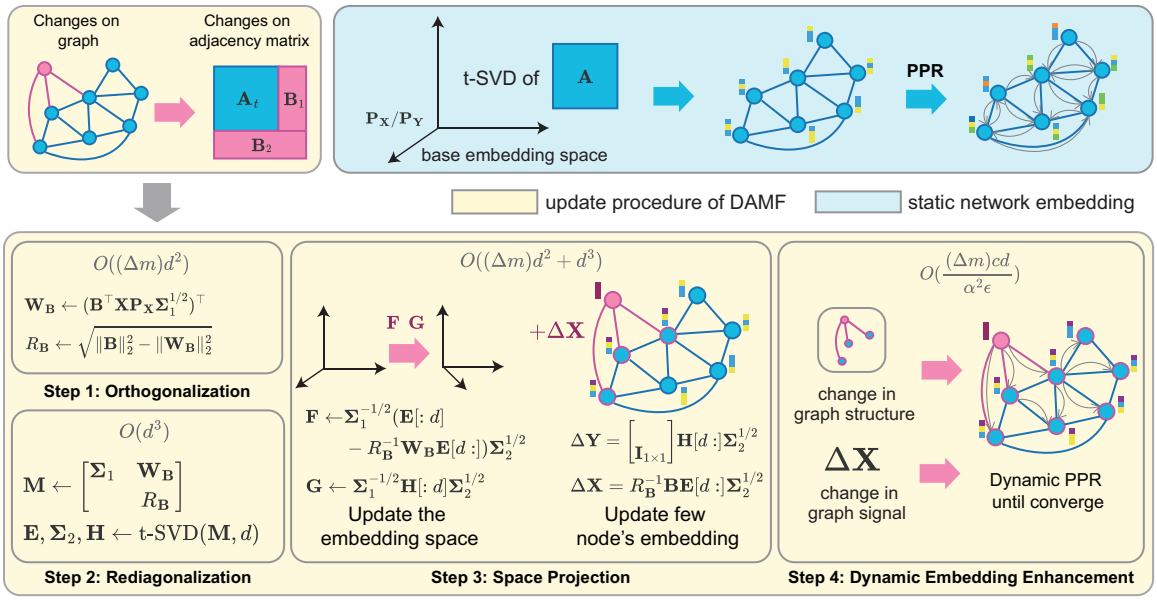

Figure 1: Overview of DAMF

The procedure for adding a row is the same, with all matrices are simply transposed.

In DAMF, without loss of generality, we consider the single node or edge update, that is, $\mathbf{B}$ (and $\mathbf{B}$, $\mathbf{C}$ in the Section 4.3) is n-by-1 matrix (or vector)

Due to the connectivity of graphs, adding nodes is often accompanied by adding edges. Let Δm be the number of edges added to the graph. Since $\mathbf{B}$ is a part of the adjacency matrix, we give the following property without proof.

Property 1. *$\mathbf{B}$ has at most Δm non-zero elements.*

For the matrix's expanded part $\mathbf{B}$, we use a space projection idea on the embedding space with a small number of modifications for node embeddings to fit the updated graph.

Lemma 1. *For arbitrary matrices $\mathbf{B} \in \mathbb{R}^{n\times q}, \mathbf{C} \in \mathbb{R}^{q\times p}$, if $\mathbf{B}$ has t non-zero rows, then $\mathbf{BC}$ has at most t non-zero rows.*

Theorem 1 (Space Projection for Node Change). *Assuming $\mathbf{B}$ is a matrix $\in \mathbb{R}^{n_1\times 1}$ with at most Δm non-zero elements. Let $\mathbf{X}_1 \in \mathbb{R}^{n_1\times d}, \mathbf{Y}_1 \in \mathbb{R}^{n_2\times d}$ be arbitrary network embedding with*

$$\mathbf{X}_1\mathbf{Y}_1^\top = \mathbf{U}_1\mathbf{\Sigma}_1\mathbf{V}_1^\top = \widetilde{\mathbf{A}}, \tag{5}$$

where $\widetilde{\mathbf{A}} \in \mathbb{R}^{n_1\times n_2}$,

then there exists space projection matrices $\mathbf{F} \in \mathbb{R}^{d\times d}, \mathbf{G} \in \mathbb{R}^{d\times d}$, and embedding modification matrices $\Delta\mathbf{X} \in \mathbb{R}^{n_1\times d}, \Delta\mathbf{Y} \in \mathbb{R}^{(n_2+1)\times d}$ with at most Δm non-zero rows , such that

$$\mathbf{X}_2 = \mathbf{X}_1\mathbf{F} + \Delta\mathbf{X}, \quad \mathbf{Y}_2 = \mathbf{Y}_1\mathbf{G} + \Delta\mathbf{Y}, \tag{6}$$

where $\mathbf{X}_2 \in \mathbb{R}^{n_1\times d}, \mathbf{Y}_2 \in \mathbb{R}^{(n_2+1)\times d}$ is a network embedding from a rank-d t-SVD of $[\widetilde{\mathbf{A}}, \mathbf{B}]$, i.e.,

$$(\mathbf{U}_2, \mathbf{\Sigma}_2, \mathbf{V}_2) \leftarrow \texttt{t-SVD}([\widetilde{\mathbf{A}}, \mathbf{B}], d) \tag{7}$$

$$\mathbf{X}_2 = \mathbf{U}_2\sqrt{\mathbf{\Sigma}_2}, \quad \mathbf{Y}_2 = \mathbf{V}_2\sqrt{\mathbf{\Sigma}_2} \tag{8}$$

Proof. Let $\mathbf{W_B} = \mathbf{U}_1^\top\mathbf{B}$, and the normalized $(\mathbf{I} - \mathbf{U}_1\mathbf{U}_1^\top)\mathbf{B}$ be

$$R_\mathbf{B} = \|(\mathbf{I} - \mathbf{U}_1\mathbf{U}_1^\top)\mathbf{B}\|_2, \quad \mathbf{Q_B} = (\mathbf{I} - \mathbf{U}_1\mathbf{U}_1^\top)\mathbf{B}/R_\mathbf{B} \tag{9}$$

It can be proved that such a $\mathbf{Q_B}$ is orthogonal to all column vectors of $\mathbf{U}_1$.

According to Zha-Simon's formula [46], we have

$$\begin{aligned} \begin{bmatrix} \mathbf{X}_1\mathbf{Y}_1^\top & \mathbf{B} \end{bmatrix} &= \begin{bmatrix} \mathbf{U}_1\mathbf{\Sigma}_1\mathbf{V}_1^\top & \mathbf{B} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{U}_1 & \mathbf{Q_B} \end{bmatrix} \begin{bmatrix} \mathbf{\Sigma}_1 & \mathbf{W_B} \\ & R_\mathbf{B} \end{bmatrix} \begin{bmatrix} \mathbf{V}_1^\top & \\ & \mathbf{I} \end{bmatrix} \\ &\approx (\begin{bmatrix} \mathbf{U}_1 & \mathbf{Q_B} \end{bmatrix} \mathbf{E})\mathbf{\Theta}(\begin{bmatrix} \mathbf{V}_1 & \\ & \mathbf{I} \end{bmatrix} \mathbf{H})^\top \\ &= \mathbf{U}_2\mathbf{\Sigma}_2\mathbf{V}_2^\top \end{aligned} \tag{10}$$

with

$$\mathbf{U}_2 = [\mathbf{U}_1 \quad \mathbf{Q}_B]\mathbf{E}, \quad \mathbf{\Sigma}_2 = \mathbf{\Theta}, \quad \mathbf{V}_2 = (\begin{bmatrix} \mathbf{V}_1 & \\ & \mathbf{I} \end{bmatrix} \mathbf{H}) \tag{11}$$

where the matrix product $\mathbf{E\Theta H}$ denotes a compact rank-d t-SVD with

$$\mathbf{E}, \mathbf{\Theta}, \mathbf{H} \leftarrow \texttt{t-SVD}(\begin{bmatrix} \mathbf{\Sigma}_1 & \mathbf{W_B} \\ & R_\mathbf{B} \end{bmatrix}, d) \tag{12}$$

What is mentioned above is Zha-Simon's t-SVD update scheme. The following will show how to obtain the space projection matrix and embedding modification vectors.

The central idea of the proof is to express the update of $\mathbf{U}, \mathbf{V}$ in Eq.(11) as a space projection onto $\mathbf{U}, \mathbf{V}$ plus a matrix with at most Δm non-zero rows. This is because in the update of $\mathbf{U}$ in Eq.(11), $\mathbf{Q_B}$ can be written as

$$\mathbf{Q_B} = R_\mathbf{B}^{-1}(\mathbf{I} - \mathbf{U}_1\mathbf{U}_1^\top)\mathbf{B} = R_\mathbf{B}^{-1}\mathbf{B} - \mathbf{U}_1(R_\mathbf{B}^{-1}\mathbf{U}_1^\top\mathbf{B}) \tag{13}$$

Notice that $R_\mathbf{B}$ is a scalar, so $R_\mathbf{B}^{-1}\mathbf{B}$ is a sparse matrix with at most Δm non-zero rows. Moreover, $\mathbf{U}_1(R_\mathbf{B}^{-1}\mathbf{U}_1^\top\mathbf{B})$ is a space projection onto $\mathbf{U}_1$ (which can be further merged).

Specifically, we split the matrix multiplication in Eq.(11) for U with

$$\begin{aligned}\mathbf{U}_2 &= [\mathbf{U}_1 \quad \mathbf{Q}_B]\mathbf{E} \\ &= \mathbf{U}_1\mathbf{E}[:d] + \mathbf{Q}_\mathbf{B}\mathbf{E}[d:] \\ &= \mathbf{U}_1\mathbf{E}[:d] + (R_\mathbf{B}^{-1}\mathbf{B} - \mathbf{U}_1(R_\mathbf{B}^{-1}\mathbf{U}_1^\top\mathbf{B})\mathbf{E}[d:] \\ &= \mathbf{U}_1(\mathbf{E}[:d] - (R_\mathbf{B}^{-1}\mathbf{U}_1^\top\mathbf{B})\mathbf{E}[d:]) + R_\mathbf{B}^{-1}\mathbf{B}\mathbf{E}[d:]\end{aligned} \tag{14}$$

which is clearly a space projection on the columns of $\mathbf{U}_1$ plus a sparse matrix with at most Δm non-zero rows. And, similarly, for the update on $\mathbf{V}$ in Eq.(11) we have

$$\mathbf{V}_2 = (\begin{bmatrix}\mathbf{V}_1 & \\ & \mathbf{I}\end{bmatrix}\mathbf{H}) = \begin{bmatrix}\mathbf{V}_1\end{bmatrix}\mathbf{H}[:\mathbf{d}] + \begin{bmatrix}\mathbf{I}_{1\times 1}\end{bmatrix}\mathbf{H}[\mathbf{d}:] \tag{15}$$

Since $\mathbf{X}_1 = \mathbf{U}_1\sqrt{\Sigma_1}$, $\mathbf{Y}_1 = \mathbf{V}_1\sqrt{\Sigma_1}$ and $\mathbf{X}_2 = \mathbf{U}_2\sqrt{\Sigma_2}$, $\mathbf{Y}_2 = \mathbf{V}_2\sqrt{\Sigma_2}$, we have

$$\begin{aligned}\mathbf{X}_2 =& \mathbf{X}_1\Sigma_1^{-1/2}(\mathbf{E}[:d] - R_\mathbf{B}^{-1}\mathbf{W}_\mathbf{B}\mathbf{E}[d:])\Sigma_2^{1/2} \\ &+ R_\mathbf{B}^{-1}\mathbf{B}\mathbf{E}[d:]\Sigma_2^{1/2} \\ \mathbf{Y}_2 =& \mathbf{Y}_1\Sigma_1^{-1/2}\mathbf{H}[:d]\Sigma_2^{1/2} \\ &+ \begin{bmatrix}\mathbf{I}_{1\times 1}\end{bmatrix}\mathbf{H}[d:]\Sigma_2^{1/2}\end{aligned} \tag{16}$$

and we take

$$\mathbf{F} = \Sigma_1^{-1/2}(\mathbf{E}[:d] - R_\mathbf{B}^{-1}\mathbf{W}_\mathbf{B}\mathbf{E}[d:])\Sigma_2^{1/2} \tag{17}$$

$$\mathbf{G} = \Sigma_1^{-1/2}\mathbf{H}[:d]\Sigma_2^{1/2} \tag{18}$$

$$\Delta\mathbf{X} = R_\mathbf{B}^{-1}\mathbf{B}\mathbf{E}[d:]\Sigma_2^{1/2} \tag{19}$$

$$\Delta\mathbf{Y} = \begin{bmatrix}\mathbf{I}_{1\times 1}\end{bmatrix}\mathbf{H}[d:]\Sigma_2^{1/2} \tag{20}$$

Since $\mathbf{B}$ has at most Δm non-zero rows (as stated in Lemma 1), $\Delta\mathbf{X}$ has at most Δm non-zero rows, and it is clear that $\Delta\mathbf{Y}$ has only one non-zero row. □

The above proof provides the space projection matrices $\mathbf{F}, \mathbf{G}$ to use in the embedding space, and shows that $\Delta\mathbf{X}$ and $\Delta\mathbf{Y}$ require at most Δm nodes' embedding to be modified additionally in a node change process.

4.3 Edge Change

Considering adding a directed edge (u, v) to the graph, the change in the adjacency matrix can be expressed as a low-rank update by

$$\mathbf{A}_t \leftarrow \mathbf{A}_{t-1} + \Delta\mathbf{A}_t, \quad \Delta\mathbf{A}_{t-1} = \mathbf{B}\mathbf{C}^\top \tag{21}$$

with

$$\mathbf{B} = e_u, \quad \mathbf{C} = e_v \tag{22}$$

where e_i denotes the standard basis (i.e., a vector whose components are all zero, except the i-th element is 1).

Theorem 2 (Space Projection for Edge Change). *Assuming $\mathbf{B}$ and $\mathbf{C}$ are matrices $\in \mathbb{R}^{n\times 1}$ with at most Δm non-zero elements. Let $\mathbf{X}_1 \in \mathbb{R}^{n\times d}, \mathbf{Y}_1 \in \mathbb{R}^{n\times d}$ be arbitrary network embedding with*

$$\mathbf{X}_1\mathbf{Y}_1^\top = \mathbf{U}_1\Sigma_1\mathbf{V}_1^\top = \widetilde{\mathbf{A}}, \tag{23}$$

where $\widetilde{\mathbf{A}} \in \mathbb{R}^{n\times n}$,

then there exists space projection matrices $\mathbf{F} \in \mathbb{R}^{d\times d}, \mathbf{G} \in \mathbb{R}^{d\times d}$, and embedding modification vectors $\Delta\mathbf{X} \in \mathbb{R}^{n\times d}$ and $\Delta\mathbf{Y} \in \mathbb{R}^{n\times d}$ with at most Δm non-zero rows , such that

$$\mathbf{X}_2 = \mathbf{X}_1\mathbf{F} + \Delta\mathbf{X}, \quad \mathbf{Y}_2 = \mathbf{Y}_1\mathbf{G} + \Delta\mathbf{Y}, \tag{24}$$

where $\mathbf{X}_2 \in \mathbb{R}^{n\times d}, \mathbf{Y}_2 \in \mathbb{R}^{n\times d}$ is a network embedding from a rank-d t-SVD of $\widetilde{\mathbf{A}} + \mathbf{B}\mathbf{C}^\top$, i.e.,

$$(\mathbf{U}_2, \Sigma_2, \mathbf{V}_2) \leftarrow \texttt{t-SVD}(\widetilde{\mathbf{A}} + \mathbf{B}\mathbf{C}^\top, d) \tag{25}$$

$$\mathbf{X}_2 = \mathbf{U}_2\sqrt{\Sigma_2}, \quad \mathbf{Y}_2 = \mathbf{V}_2\sqrt{\Sigma_2} \tag{26}$$

Proof. Similar to the proof of Theorem 1, firstly, let $\mathbf{W}_\mathbf{B} = \mathbf{U}_1^\top B$ and $\mathbf{W}_\mathbf{C} = \mathbf{V}_1^\top\mathbf{C}$. And the normalized $(\mathbf{I} - \mathbf{U}_1\mathbf{U}_1^\top)\mathbf{B}$, $(\mathbf{I} - \mathbf{V}_1\mathbf{V}_1^\top)\mathbf{C}$ can be calculated by

$$R_\mathbf{B} = \|(\mathbf{I} - \mathbf{U}_1\mathbf{U}_1^\top)\mathbf{B}\|_2, \quad \mathbf{Q}_\mathbf{B} = (\mathbf{I} - \mathbf{U}_1\mathbf{U}_1^\top)\mathbf{B}/R_\mathbf{B} \tag{27}$$

$$R_\mathbf{C} = \|(\mathbf{I} - \mathbf{V}_1\mathbf{V}_1^\top)\mathbf{C}\|_2, \quad \mathbf{Q}_\mathbf{C} = (\mathbf{I} - \mathbf{V}_1\mathbf{V}_1^\top)\mathbf{C}/R_\mathbf{C} \tag{28}$$

Then let

$$\mathbf{E}, \Sigma_2, \mathbf{H} \leftarrow \texttt{t-SVD}(\begin{bmatrix}\Sigma_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0}\end{bmatrix} + \begin{bmatrix}\mathbf{W}_\mathbf{B} \\ R_\mathbf{B}\end{bmatrix}\begin{bmatrix}\mathbf{W}_\mathbf{C} \\ R_\mathbf{C}\end{bmatrix}^\top, d) \tag{29}$$

We can get the space projection matrices $\mathbf{F}, \mathbf{G}$ and embedding modification vectors $\Delta\mathbf{X}, \Delta\mathbf{Y}$ by

$$\mathbf{F} = \Sigma_1^{-1/2}(\mathbf{E}[:d] - R_\mathbf{B}^{-1}\mathbf{W}_\mathbf{B}\mathbf{E}[d:])\Sigma_2^{1/2} \tag{30}$$

$$\mathbf{G} = \Sigma_1^{-1/2}(\mathbf{H}[:d] - R_\mathbf{C}^{-1}\mathbf{W}_\mathbf{C}\mathbf{H}[d:])\Sigma_2^{1/2} \tag{31}$$

$$\Delta\mathbf{X} = R_\mathbf{B}^{-1}\mathbf{B}\mathbf{E}[d:]\Sigma_2^{1/2} \tag{32}$$

$$\Delta\mathbf{Y} = R_\mathbf{C}^{-1}\mathbf{C}\mathbf{H}[d:]\Sigma_2^{1/2} \tag{33}$$

Since $\mathbf{B}, \mathbf{C}$ have at most Δm non-zero rows, by Lemma 1, $\Delta\mathbf{X}, \Delta\mathbf{Y}$ have at most Δm non-zero rows. □

The above proof provides the space projection matrices $\mathbf{F}, \mathbf{G}$ to use in the embedding space, and shows that $\Delta\mathbf{X}$ and $\Delta\mathbf{Y}$ require at most Δm nodes' embedding to be modified additionally in an edge change process.

4.4 Dynamic Embedding Enhancement via PPR

In order to capture higher-order neighbors's information, an enhancement is applied to the dynamic network embedding. Specifically, we apply a dynamic *Personalized PageRank* (PPR)[26] to the updated context embedding $\mathbf{X}$ to get the enhanced context embedding $\mathbf{Z}$.

PPR Enhancement in Static Network Embedding. To better capture higher-order neighborhood information, a mainstream approach on static network embeddings is to use *Personalized PageRank* (PPR) [26] to enhance the network embeddings (e.g., APP, Lemane, STRAP, VERSE, and NRP).

Specifically, for a graph with adjacency matrix $\mathbf{A}$ and graph signal $\mathbf{X} \in \mathbb{R}^{n\times d}$, the principles of PPR can be formulated as

$$\mathsf{PPR}(\mathbf{X}) = \sum_{i=0}^{\infty}\alpha(1-\alpha)^i(\mathbf{D}^{-1}\mathbf{A})^i\mathbf{X} \tag{34}$$

where $\mathbf{D} \in \mathbb{R}^{n\times n}$ is the diagonal out-degree matrix and α is the damping factor for PageRank.

Since solving the PPR is an infinite process, the truncated version of PPR is commonly used in practice. Moreover, to better control errors and balance efficiency, existing PageRank-based static network

embedding methods (e.g., all methods mentioned above) typically introduce an error tolerance ϵ and require PPR to converge to that error.

Dynamic Embedding Enhancement via Dynamic PPR. Unfortunately, despite the success of PageRank enhancements in static network embeddings, their methods cannot be directly migrated to dynamic scenarios. In this work, following [3, 39, 47, 52], we approximate the PPR based on graph propagation under a given error limit ϵ.

Specifically, we use an InstantGNN [52] to enhance the embedding obtained in Section 4.2 and Section 4.3. InstantGNN is an efficient dynamic PPR method suitable for graph structure changes and node signals (attributes). In Dynamic Embedding Enhancement, we use the embedding obtained from dynamic truncated singular value decomposition as the graph signal input to InstantGNN. Moreover, as the network changes, according to Theorem 1 and Theorem 2, at most Δm nodes' signals need to be changed. The InstantGNN updates the result, which converges to an error tolerance ϵ according to the graph structure and node signal change to achieve the Dynamic Embedding Enhancement.

4.5 the Proposed DAMF Algorithm

In this section, we propose the Dynamic Adjacency Matrix Factorization (DAMF) algorithm, consisting of four steps: Orthogonalization, Rediagonalization, Space Projection, and Dynamic Embedding Enhancement. Algorithm 1 and Algorithm 2 is the pseudo-code for the DAMF algorithm.

Step 1: Orthogonalization. According to the equations $\mathbf{X} = \mathbf{U}\sqrt{\Sigma}$ and $\mathbf{X}_t = \mathbf{X}_b\mathbf{P}_\mathbf{X}$, we know that $\mathbf{U} = \mathbf{X}_b\mathbf{P}_\mathbf{X}\Sigma^{-1/2}$. Then, with $\mathbf{W_B} = \mathbf{U}^\top\mathbf{B}$ (and $\mathbf{W_C} = \mathbf{V}^\top\mathbf{C}$ for edge change), we can get that $R_B = \sqrt{\|\mathbf{B}\|_2^2 - \|\mathbf{W_B}\|_2^2}$, (and $R_C = \sqrt{\|\mathbf{C}\|_2^2 - \|\mathbf{W_C}\|_2^2}$ for edge change) by Eq. (9) and Eq. (28) and Proposition 1.

Proposition 1. *Let $\mathbf{U} \in \mathbb{R}^{n\times d}$ be an arbitrary orthonormal matrix with $\mathbf{U}^\top\mathbf{U} = \mathbf{I}$. and $\vec{x} \in \mathbb{R}^n$ be an arbitrary vector, then*

$$\|(\mathbf{I} - \mathbf{U}\mathbf{U}^\top)\vec{x}\|_2 = \sqrt{\|\vec{x}\|_2^2 - \|\mathbf{U}^\top\vec{x}\|_2^2}$$

Step 2: Rediagonalization. In this step, follow Eq. (12) for node change, and Eq. (29) for edge change to get the t-SVD.

Step 3: Space Rotation. For node change, get $\mathbf{F}, \mathbf{G}, \Delta\mathbf{X}, \Delta\mathbf{Y}$ by Eq. (17), Eq. (18), Eq. (19), Eq. (20), respectively. And for edge change, get $\mathbf{F}, \mathbf{G}, \Delta\mathbf{X}, \Delta\mathbf{Y}$ by Eq. (30), Eq. (31), Eq. (32), Eq. (33), respectively. Then update the network embedding with space projection matrix $\mathbf{F}, \mathbf{G}, \Delta\mathbf{X}, \Delta\mathbf{Y}$ by Eq. (3).

Step 4: Dynamic Embedding Enhancement. In this step, the PPR-enhanced embedding is updated by InstantGNN [52]. Specifically, InstantGNN can estimate the PPR values for dynamic graph structures and dynamic node attributes. Here, we treat the changes in the graph as the dynamic graph structure, and the updates of few nodes' embedding as the dynamic node attributes. InstantGNN will update the embedding so that the results converge to the given error tolerance.

Initialization of DAMF. Initially, $\mathbf{X}_b, \mathbf{Y}_b$ is set by a t-SVD of the adjacency matrix $\mathbf{A}$ of the initial graph, and $\mathbf{P_X}, \mathbf{P_Y}$ is set to the identity matrix. Then, use the basic propagation of PPR in InstantGNN [52] to enhance the initial network embedding.

Algorithm 1: Update embedding via space projection

Procedure UpdateEmbeddingN($\mathbf{X}, \mathbf{Y}, \mathbf{P_X}, \mathbf{P_Y}, \Sigma_1, \mathbf{B}, d$)
- /* Step 1: Orthogonalization */
- $\mathbf{W_B} \leftarrow (\mathbf{B}^\top\mathbf{X}\mathbf{P_X}\Sigma_1^{1/2})^\top$;
- $R_\mathbf{B} \leftarrow \sqrt{\|\mathbf{B}\|_2^2 - \|\mathbf{W_B}\|_2^2}$;
- /* Step 2: Rediagonalization */
- $\mathbf{M} \leftarrow \begin{bmatrix}\Sigma_1 & \mathbf{W_B}\\ & R_\mathbf{B}\end{bmatrix}$; // $\mathbf{M} \in \mathbb{R}^{d+1,d+1}$
- $\mathbf{E}, \Sigma_2, \mathbf{H} \leftarrow$ t-SVD($\mathbf{M}, d$) ;
- /* Step 3: Space Projection */
- $\mathbf{F} \leftarrow \Sigma_1^{-1/2}(\mathbf{E}[:d] - R_\mathbf{B}^{-1}\mathbf{W_B}\mathbf{E}[d:])\Sigma_2^{1/2}$;
- $\mathbf{G} \leftarrow \Sigma_1^{-1/2}\mathbf{H}[:d]\Sigma_2^{1/2}$;
- $\Delta\mathbf{X} = R_\mathbf{B}^{-1}\mathbf{B}\mathbf{E}[d:]\Sigma_2^{1/2}$;
- $\Delta\mathbf{Y} = \begin{bmatrix}\\ \mathbf{I}_{1\times 1}\end{bmatrix}\mathbf{H}[d:]\Sigma_2^{1/2}$;
- **return** $\mathbf{F}, \mathbf{G}, \Delta\mathbf{X}, \Delta\mathbf{Y}, \Sigma_2$;

Procedure UpdateEmbeddingE($\mathbf{X}, \mathbf{Y}, \mathbf{P_X}, \mathbf{P_Y}, \Sigma_1, \mathbf{B}, \mathbf{C}, d$)
- /* Step 1: Orthogonalization */
- $\mathbf{W_B} \leftarrow (\mathbf{B}^\top\mathbf{X}\mathbf{P_X}\Sigma_1^{1/2})^\top$;
- $\mathbf{W_C} \leftarrow (\mathbf{C}^\top\mathbf{Y}\mathbf{P_Y}\Sigma_1^{1/2})^\top$;
- $R_\mathbf{B} \leftarrow \sqrt{\|\mathbf{B}\|_2^2 - \|\mathbf{W_B}\|_2^2}$;
- $R_\mathbf{C} \leftarrow \sqrt{\|\mathbf{C}\|_2^2 - \|\mathbf{W_C}\|_2^2}$;
- /* Step 2: Rediagonalization */
- $\mathbf{M} \leftarrow \begin{bmatrix}\Sigma_1 & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{bmatrix} + \begin{bmatrix}\mathbf{W_B}\\ \mathbf{R_B}\end{bmatrix}\begin{bmatrix}\mathbf{W_C}\\ \mathbf{R_C}\end{bmatrix}^\top$;
- $\mathbf{E}, \Sigma_2, \mathbf{H} \leftarrow$ t-SVD($\mathbf{M}, d$) ;
- /* Step 3: Space Projection */
- $\mathbf{F} \leftarrow \Sigma_1^{-1/2}(\mathbf{E}[:d] - R_\mathbf{B}^{-1}\mathbf{W_B}\mathbf{E}[d:])\Sigma_2^{\frac{1}{2}}$;
- $\mathbf{G} \leftarrow \Sigma_1^{-1/2}(\mathbf{H}[:d] - R_\mathbf{C}^{-1}\mathbf{W_C}\mathbf{H}[d:])\Sigma_2^{1/2}$;
- $\Delta\mathbf{X} \leftarrow R_\mathbf{B}^{-1}\mathbf{B}\mathbf{E}[d:]\Sigma_2^{1/2}$;
- $\Delta\mathbf{Y} \leftarrow R_\mathbf{C}^{-1}\mathbf{C}\mathbf{H}[d:]\Sigma_2^{1/2}$;
- **return** $\mathbf{F}, \mathbf{G}, \Delta\mathbf{X}, \Delta\mathbf{Y}, \Sigma_2$;

4.6 Complexity Analysis

In this section, we analyze the DAMF algorithm in terms of time and space complexity. Note that the time complexity we analysis here is only for a single graph change.

DAMF is very efficient due to the fact that the large-scale matrix multiplications involved in DAMF are mainly of two kinds showed in Figure 2. And Lemma 2 (Figure 2(a)) and Lemma 3 ((Figure 2(b))) demonstrate how to use sparsity of these two special matrix multiplications and how they can be efficiently computed. The proofs of these two lemmas are in the Appendix B.3 and Appendix B.4.

Lemma 2. *Let $\mathbf{A} \in \mathbb{R}^{n\times p}, \mathbf{B} \in \mathbb{R}^{n\times q}$ be arbitrary matrices with $\mathbf{B}$ has t non-zero rows, the time complexity to calculate $\mathbf{B}^\top\mathbf{A}$ is $O(tpq)$.*

Lemma 3. *Let $\mathbf{B} \in \mathbb{R}^{n\times q}, \mathbf{C} \in \mathbb{R}^{q\times p}$ be arbitrary matrices with $\mathbf{B}$ has t non-zero rows, the time complexity to calculate $\mathbf{BC}$ is $O(tpq)$.*

Algorithm 2: DAMF

Input: Network embedding $\mathbf{X}_b, \mathbf{Y}_b$ in the base space; enhanced embedding $\mathbf{Z}_b$; space projection matrix $\mathbf{P_X}, \mathbf{P_Y}$; singular values Σ; the Changes occurring in the network $Event_t$ residual vector $\mathbf{r}$; PageRank damping factor α; error tolerance ϵ; Graph $\mathcal{G}$;

Output: Updated $\mathbf{X}_b, \mathbf{Y}_b, \mathbf{Z}_b, \mathbf{P_X}, \mathbf{P_Y}, \Sigma$

1 **if** $Event_t$ *is node change* **then**
2 Convert $Event_t$ to $\mathbf{B}_1, \mathbf{B}_2$ by Eq. (4) ;
3 $\mathbf{F}, \mathbf{G}, \Delta\mathbf{X}, \Delta\mathbf{Y}, \Sigma \leftarrow$ `UpdateEmebddingN`$(\mathbf{X}, \mathbf{Y}, \Sigma, \mathbf{B}_1)$;
4 Update $\mathbf{X}_b, \mathbf{Y}_b, \mathbf{P_X}, \mathbf{P_Y}$ by Eq. (3) ;
5 $\mathbf{F}, \mathbf{G}, \Delta\mathbf{X}, \Delta\mathbf{Y}, \Sigma \leftarrow$ `UpdateEmebddingN`$(\mathbf{Y}, \mathbf{X}, \Sigma, \mathbf{B}_2)$;
6 Update $\mathbf{X}_b, \mathbf{Y}_b, \mathbf{P_X}, \mathbf{P_Y}$ by Eq. (3) ;
7 **else**
8 Convert $Event_t$ to $\mathbf{B}, \mathbf{C}$ by Eq. (22) ;
9 $\mathbf{F}, \mathbf{G}, \Delta\mathbf{X}, \Delta\mathbf{Y}, \Sigma \leftarrow$ `UpdateEmbeddingM`$(\mathbf{X}, \mathbf{Y}, \Sigma, \mathbf{B}, \mathbf{C})$;
10 Update $\mathbf{X}_b, \mathbf{Y}_b, \mathbf{P_X}, \mathbf{P_Y}$ by Eq. (3) ;
11 **end**
12 $\mathbf{Z}_b, \mathbf{r} \leftarrow$ `DynamicEnhancement`$(\mathbf{Z}_b, \mathbf{r}, Event_t, \mathcal{G}, \Delta\mathbf{X}, \alpha, \epsilon)$;
13 **return** $\mathbf{X}_b, \mathbf{Y}_b, \mathbf{Z}_b, \mathbf{P_X}, \mathbf{P_Y}, \Sigma$

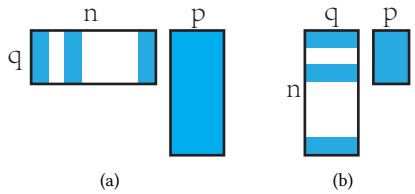

Figure 2: Two special matrix multiplications that can be efficiently computed as proved by Lemma 2 and Lemma 3 (the white color indicates zero elements in matrices)

Time Complexity of Orthogonalization (Step 1). According to Lemma 2, the time complexity of calculating $\mathbf{B}^\top\mathbf{X}$ (or $\mathbf{C}^\top\mathbf{Y}$) is $O((\Delta m)d^2)$ since $\mathbf{B}$ (and $\mathbf{C}$) has at most Δm non-zero rows. Next, since $\mathbf{P_X}$ and $\Sigma_1^{1/2}$ are d-by-d matrices, the complexity of calculating $\mathbf{W_B}$ (or $\mathbf{W_C}$, Line 2, 12, 13 in Algorithm 1) is $O((\Delta m)d^2)$. At last, the time complexity of calculating $R_\mathbf{B}$(or $R_\mathbf{C}$, Line 3, 14, 15 in Algorithm 1) is $O(\Delta m + d)$. Thus, the overall time complexity of the Orthogonalization step is $O((\Delta m)d^2)$.

Time Complexity of Rediagonalization (Step 2). The time complexity of the Rediagonalization step is $O(d^3)$ by directly applying a t-SVD on a $(d+1)$-by-$(d+1)$ matrix (Line 4, 5, 16, 17 in Algorithm 1).

Time Complexity of Space Rotation (Step 3). The time complexity of calculating $\mathbf{F}$ and $\mathbf{G}$ (Line 6, 7, 18, 19 in Algorithm 1) is $O(d^3)$. According to Lemma 3, the time complexity of calculating $\Delta\mathbf{X}$ and $\Delta\mathbf{Y}$ (Line 8, 9, 20, 21 in Algorithm 1) is $O((\Delta m)d^2)$.

Moreover, the time complexity of space projection (Line 4, 6, 10 in Algorithm 2) is $O((\Delta m)d^2 + d^3)$ by the following steps: (1) computing the update of $\mathbf{P_X}$ by a d-by-d matrix multiplication; (2) computing the inverse of $\mathbf{P_X}$; (3) computing $\mathbf{B}\mathbf{P}_\mathbf{X}^{-1}$. The time complexity of both the matrix multiplication and the inverse is $O(d^3)$, while the time complexity of $\mathbf{B}\mathbf{P}_\mathbf{X}^{-1}$ is $O((\Delta m)d^2)$ according to Lemma 3.

Table 2: Time Complexity of DAMF

	Name	Time Complexity
Step 1	Orthogonalization	$O((\Delta m)d^2)$
Step 2	Rediagonalization	$O(d^3)$
Step 3	Space Rotation	$O((\Delta m)d^2 + d^3)$
Step 4	Dynamic Embedding Enhancement	$O(\frac{(\Delta m)cd}{\alpha^2\epsilon})$

Time Complexity of Dynamic Embedding Enhancement (Step 4). Let $c = \max_{u\in\mathcal{V}} \|deg(u)\mathbf{X}(u)\|_\infty$ where $\mathbf{X}[u]$ is the node u's context embedding. Since the number of edges changed by the graph structure is Δm, and at most Δm nodes' context embedding that need to be updated in the previous step, according to Theorem 3 and Theorem 4 in [52], the time complexity of this step is $O(\frac{(\Delta m)cd}{\alpha^2\epsilon})$.

Overall, if we consider embedding dimension d, Personalized PageRank damping factor α, error tolerance ϵ and c as constant, the purposed DAMF algorithm achieves a dynamic network embedding for a single graph change with time complexity $O(\Delta m)$.

Time Complexity of Query Node's Embedding The node embedding query at any moment can be obtained by multiplying the embedding $\mathbf{X}_b, \mathbf{Y}_b$ of that node in the initial space by the space projection matrix $\mathbf{P_X}, \mathbf{P_y}$ with time complexity of $O(d^2)$, while the direct query in the static embedding method has a time complexity of $O(d)$. When we consider dimensionality a constant, the query complexity is $O(1)$.

Space Complexity. The largest matrices in the DAMF method is an n-by-d matrix, so the space complexity is $O(nd)$. Due to the additional need to store the structure of the graph, the total space complexity is $O(nd + m)$.

Table 3: Statistics of Datasets

Dataset		$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	#labels
small	Wiki	4,777	184,812	40
	Cora	12,022	45,421	10
	Flickr	80,513	11,799,764	195
large	YouTube	1,138,499	2,990,443	47
	Orkut	3,072,441	117,185,083	100
massive	Twitter	41,652,230	**1,468,365,182**	-

5 EXPERIMENT

In this section, we experimentally compare DAMF to six existing methods on six datasets of varying sizes for three popular analytic tasks: node classification, link prediction, and graph reconstruction. Section 5.1 and Section 5.2 introduce the experimental settings and tasks. In Section 5.3, Section 5.3, and Section 5.3, we present our experimental results on small, large, and massive graphs respectively. Finally, we compare the runtime of DAMF with baselines in Section 5.4 and analyze the results of the ablation study on dynamic embedding enhancement in Section 5.5.

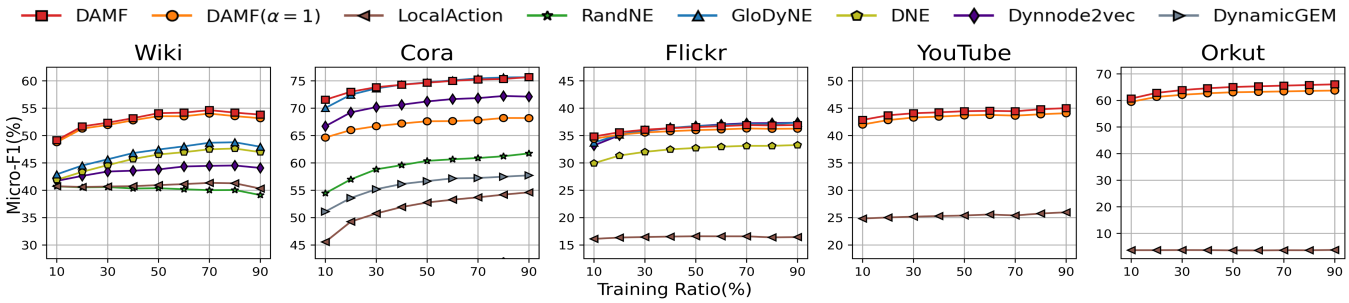

Figure 3: Node classification's predictive performance w.r.t. the ratio of training data

5.1 Experimental Settings

Baseline. We evaluate DAMF against six existing methods, including LocalAction [19], GloDyNE [15], Dynnode2vec [21], RandNE [49], DynGEM [11] and DNE [8].

Datasets. We conduct experiments on 6 publicly available graph datasets and divide the datasets into small, large, and massive scales. As shown in Table 3, small ones include *Wiki* [22], *Cora* [15] and *Flickr* [36], large ones include *YouTube* [37] and *Orkut* [23], and *Twitter* [17] is considered as a massive dataset. A short description of each dataset is given in Appendix C.3.

Setting of Dynamicity. We follow GloDyNE's setup for the real dataset *Cora* [15], a widely accepted dataset whose dynamic graph has 11 snapshots. For other datasets, following the setting in LocalAction [19] and DNE [8], we start with a small number of nodes and gradually add the remaining ones to the graph individually (streaming scenario); for the discrete methods GloDyNE [15], dynGEM [11], Dynnode2vec [21], the continuous streaming modifications are split into 100 discrete modifications. To reflect the reality of the recent exponential rise in social network users, we utilize a considerably lower initial setting of 1000 nodes compared with the methods above.

Time Limits. For small and large datasets, methods that have not produced results for more than 3 days (72 hours) will not be included in the results. For massive datasets, methods that have not produced results for more than 7 days (168 hours) will not be included in the results.

Paramater Settings. The embedding dimension d of all methods is set to 128 for a fair comparison. For DAMF, the damping factor α for PPR is set to 0.3 (except for the node classification task on *Cora* where it is set to 0.03) and the error tolerance ϵ is set to 10^{-5}.

5.2 Experimental Tasks

Dynamic network embedding is tested on three tasks: node classification, link prediction and graph reconstruction.

Node Classification is a common model training task to obtain the labels based on the embedding of each node by training a simple classifier. Following previous work [38, 42], we randomly select a subset of nodes to train a one-vs-all logistic regression classifier, and then use the classifier to predict the labels of the remaining nodes. For each node v, we first concatenate the normalized context and content vectors as the feature representation of v and then feed it to the classifier.

Link Prediction is the method of predicting the possibility of a link between two graph nodes. Based on earlier research, we first generate the modified graph $\mathcal{G}'$ by removing 30% of randomly selected edges from the input graph $\mathcal{G}$, and then construct embeddings on $\mathcal{G}'$. Then, we build the testing set $\mathcal{E}_{test}$ by selecting the node pairs connected by the removed edges and an equal number of unconnected node pairs in $\mathcal{G}$.

The inner product of node's embedding are used to make predictions. Results are evaluated by *Area Under Curve (AUC)* and *Average Precision(AP)*.

Graph Reconstruction is a basic objective of network embedding. According to earlier work, we begin by selecting a set $\mathcal{S}$ of node pairs from the input graph $\mathcal{G}$. Afterward, we apply the same method used in link prediction to generate a score for each pair. Then, we calculate the *precision@K*, the proportion of the top-K node pairings that match edges in $\mathcal{G}$.

Table 4: Link prediction results on small graphs
TLE: Time Limit Exceeded, ×: No Legal Output

Method \ Dataset	AUC			AP		
	Wiki	Cora	Flickr	Wiki	Cora	Flickr
LocalAction	0.5083	0.5562	0.4995	0.5812	0.5923	0.5833
GloDyNE	0.6386	**0.9296**	0.8511	0.6844	**0.9383**	0.8639
Dynnode2vec	0.5904	0.8242	0.8187	0.6639	0.8975	0.8476
RandNE	0.1092	0.8324	TLE	0.3208	0.8435	TLE
DynGEM	×	0.8340	TLE	×	0.8568	TLE
DNE	0.1845	0.8407	0.5361	0.3391	0.8662	0.5287
DAMF(α = 1)	0.6618	0.8555	0.9269	0.7541	0.8853	0.9474
DAMF	**0.7543**	0.9010	**0.9550**	**0.7840**	0.9168	**0.9615**

5.3 Experiment

Experiment on Small Graphs. We conduct experiments on small graphs *Wiki*, *Cora* and *Flickr* with all three tasks of Node Classification, Link Prediction, and Graph Reconstruction.

The results of node classification, link prediction and graph reconstruction are displayed in Figure 3, Table 4 and Figure 4 respectively. On the node classification task, DAMF outperforms all other competitors significantly at *Wiki*, is comparable to other methods on other datasets, and had better performance when the ratio of training data was smaller. For link prediction, we can see that DAMF outperforms the baselines on almost all datasets based on both *AUC* and *AP* values, except on the *Cora* dataset where it is slightly inferior compared to GloDyNE. In terms of the result of

graph reconstruction, DAMF is significantly better than the other baselines. Overall, the DAMF method performs well and is stable across different data sets.

Table 5: Link prediction results on large and massive graphs

Method \ Dataset	AUC			AP		
	Youtube	Orkut	Twitter	Youtube	Orkut	Twitter
LocalAction	0.4849	0.4978	0.5006	0.5548	0.5355	0.5923
DAMF($\alpha = 1$)	0.7648	0.8662	0.8732	0.8290	0.8828	0.9018
DAMF	**0.7946**	**0.8724**	**0.9055**	**0.8510**	**0.8882**	**0.9353**

Experiment on Large Graphs. Our extensive graph includes *YouTube* and *Orkut* datasets. Unlike small graphs, we only conduct Node Classification and Link Prediction. Due to the high number of potential $\binom{n}{2}$ pairs, we discard the Graph Reconstruction task. It is worth noting that only LocalAction reaches the set time limit among all competitors.

The results of node classification presented in Figure 3 indicate that DAMF performs significantly better in comparison to LocalAction on both large graphs. The *AUC* and *AP* results for the link prediction task are shown in Table 5. In both indicators, DAMF has a significant advantage over the only competitor that can complete the work within the allotted time on large graphs.

Experiment on the Massive Graphs. Massive graphs with billion-level edges such as large-scale social networks are widely available in the industry. However, to the best of our knowledge, dynamic network embedding studies on massive graphs with billion-level edges are unprecedented. We conduct the first dynamic network embedding experiments on a billion-level edge dataset *Twitter* with 41 million nodes and **1.4 billion** edges, and mapping each node as a 128-dimensional vector which is more than 10 times as many learnable parameters as BERT-Large [16].

We conduct a link prediction task to determine the nodes' connection probability. Notably, we reduce the ratio of edges deleted in the first stage from 30% to 0.5% to ensure graph connectivity when there are only a few nodes. We discard node classification and graph reconstruction tasks since the Twitter dataset lacks labels to build a classifier and the excessive number of potential pairings makes reconstruction unattainable.

Table 5 shows the *AUC* and *AP* values. Overall, DAMF performs well on both evaluation indicators, obtaining an *AUC* value of 0.9055 and an *AP* of 0.9353. Moreover, DAMF's overall updating time of **110** hours demonstrates that DAMF is capable of changing parameters at the billion level in under 10 milliseconds.

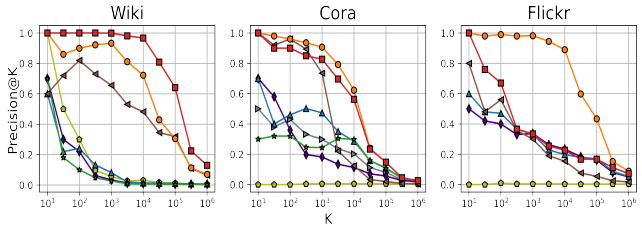

Figure 4: Graph Reconstruction (*precision@K*)

Table 6: Running time

Method \ Size	Small			Large	
	Wiki	Cora	Flickr	Youtube	Orkut
LocalAction	**7s**	**6s**	**5m28s**	**5m10s**	**1h43m**
GloDyNE	**10m45s**	**1m5s**	**18h38m**	>3days	>3days
Dynnode2vec	**1m58s**	**1m40s**	**5h10m**	>3days	>3days
RandNE	**3m**	**5s**	>3days	>3days	>3days
DynGEM	**17h4m**	**6h20m**	>3days	>3days	>3days
DNE	**22m38s**	**8m57s**	**8h11m**	>3days	>3days
DAMF($\alpha = 1$)	**36s**	**1m19s**	**33m13s**	**3h10m**	**8h15m**
DAMF	**2m47s**	**2m6s**	**1h33m**	**3h39m**	**14h7m**

All baseline methods except LocalAction [19] exceed the set time limit (7 days) and are therefore excluded. However, LocalAction seems to fail to converge on this dataset since its *AUC* on link prediction is only 0.5006.

5.4 Efficency Study

Table 6 shows the running time of each method on small and large datasets. The experimental results show that the speed of DAMF increases more consistently as the size of the dataset expands. According to Table 4 and Table 5, although LocalAction is faster, its *AUC* is only slightly greater than 0.5, indicating that LocalAction makes unbearable tread-offs to obtain speed, whereas DAMF maintains stable effectiveness.

5.5 Ablation Study

To investigate whether the dynamic embedding enhancement better captures the information of higher-order neighbors and thus improves the quality of dynamic network embedding, we obtain unenhanced embeddings by setting the PageRank damping factor α to 1. The experimental results in Figure 3 ,Table 4, Table 5 and Figure 4 show that the enhanced embeddings are significantly better in node classification, link prediction and graph reconstruction on *Wiki*, demonstrating the effectiveness of the dynamic embedding enhancement.

6 CONCLUSION

In this work, we propose the Dynamic Adjacency Matrix Factorization (DAMF) algorithm, which utilizes a projection onto the embedding space and modifies a few node embeddings to achieve an efficient adjacency matrix decomposition method. In addition, we use dynamic Personalized PageRank to enhance the embedding to capture high-order neighbors' information dynamically. Experimental results show that our proposed method performs well in node classification, link prediction, and graph reconstruction, achieving an average dynamic network embedding update time of 10ms on billion-edge graphs.

Acknowledgement. This work is supported by NSFC (No.62176233), National Key Research and Development Project of China (No.2018AAA0101900) and Fundamental Research Funds for the Central Universities.

REFERENCES

[1] Sami Abu-El-Haija, Bryan Perozzi, Rami Al-Rfou, and Alexander A Alemi. 2018. Watch your step: Learning node embeddings via graph attention. *Advances in neural information processing systems* 31 (2018).

[2] Nino Arsov and Georgina Mirceva. 2019. Network Embedding: An Overview. *ArXiv* abs/1911.11726 (2019).

[3] Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. 2020. Scaling graph neural networks with approximate pagerank. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2464–2473.

[4] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2016. Deep neural networks for learning graph representations. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 30.

[5] Chen Chen and Hanghang Tong. 2015. Fast eigen-functions tracking on dynamic graphs. In *Proceedings of the 2015 SIAM international conference on data mining*. SIAM, 559–567.

[6] Shivani Choudhary, Tarun Luthra, Ashima Mittal, and Rajat Singh. 2021. A Survey of Knowledge Graph Embedding and Their Applications. *ArXiv* abs/2107.07842 (2021).

[7] 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.

[8] 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.

[9] Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997* (2018).

[10] Maoguo Gong, Shunfei Ji, Yu Xie, Yuan Gao, and AK Qin. 2020. Exploring temporal information for dynamic network embedding. *IEEE Transactions on Knowledge and Data Engineering* 34, 8 (2020), 3754–3764.

[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] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. 2011. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review* 53, 2 (2011), 217–288.

[14] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584* (2017).

[15] Chengbin Hou, Han Zhang, Shan He, and Ke Tang. 2020. Glodyne: Global topology preserving dynamic network embedding. *IEEE Transactions on Knowledge and Data Engineering* 34, 10 (2020), 4826–4837.

[16] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of NAACL-HLT*. 4171–4186.

[17] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. 2010. What is Twitter, a social network or a news media?. In *Proceedings of the 19th international conference on World wide web*. 591–600.

[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] Xi Liu, Ping-Chun Hsieh, Nick Duffield, Rui Chen, Muhe Xie, and Xidao Wen. 2019. Real-time streaming graph embedding through local actions. In *Companion Proceedings of The 2019 World Wide Web Conference*. 285–293.

[20] Jianxin Ma, Peng Cui, and Wenwu Zhu. 2018. Depthlgp: Learning embeddings of out-of-sample nodes in dynamic networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[21] 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.

[22] Matt Mahoney. 2011. Large text compression benchmark.

[23] Alan Mislove, Massimiliano Marcon, Krishna P Gummadi, Peter Druschel, and Bobby Bhattacharjee. 2007. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. 29–42.

[24] Walter Nelson, Marinka Zitnik, Bo Wang, Jure Leskovec, Anna Goldenberg, and Roded Sharan. 2019. To embed or not: network embedding as a paradigm in computational biology. *Frontiers in genetics* 10 (2019), 381.

[25] 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.

[26] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank citation ranking: Bringing order to the web.* Technical Report. Stanford InfoLab.

[27] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. 2020. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 5363–5370.

[28] 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.

[29] Jiezhong Qiu, Laxman Dhulipala, Jie Tang, Richard Peng, and Chi Wang. 2021. Lightne: A lightweight graph processing system for network embedding. In *Proceedings of the 2021 international conference on management of data*. 2281–2289.

[30] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Chi Wang, Kuansan Wang, and Jie Tang. 2019. Netsmf: Large-scale network embedding as sparse matrix factorization. In *The World Wide Web Conference*. 1509–1520.

[31] 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.

[32] Chuan Shi, Binbin Hu, Wayne Xin Zhao, and S Yu Philip. 2018. Heterogeneous information network embedding for recommendation. *IEEE Transactions on Knowledge and Data Engineering* 31, 2 (2018), 357–370.

[33] Uriel Singer, Ido Guy, and Kira Radinsky. 2019. Node embedding over temporal graphs. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. 4605–4612.

[34] Chang Su, Jie Tong, Yongjun Zhu, Peng Cui, and Fei Wang. 2020. Network embedding in biomedical data science. *Briefings in bioinformatics* 21, 1 (2020), 182–197.

[35] 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.

[36] Lei Tang and Huan Liu. 2009. Relational learning via latent social dimensions. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. 817–826.

[37] Lei Tang and Huan Liu. 2009. Scalable learning of collective behavior based on sparse social dimensions. In *Proceedings of the 18th ACM conference on Information and knowledge management*. 1107–1116.

[38] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. 2018. Verse: Versatile graph embeddings from similarity measures. In *Proceedings of the 2018 world wide web conference*. 539–548.

[39] Hanzhi Wang, Mingguo He, Zhewei Wei, Sibo Wang, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. 2021. Approximate graph propagation. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 1686–1696.

[40] Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo. 2018. Graphgan: Graph representation learning with generative adversarial nets. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

[41] Yufei Wen, Lei Guo, Zhumin Chen, and Jun Ma. 2018. Network embedding based recommendation method in social networks. In *Companion Proceedings of the The Web Conference 2018*. 11–12.

[42] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, and Sourav S Bhowmick. 2020. Homogeneous network embedding for massive graphs via reweighted personalized PageRank. *Proceedings of the VLDB Endowment* 13, 5 (2020), 670–683.

[43] Yuan Yin and Zhewei Wei. 2019. Scalable graph embeddings via sparse transpose proximities. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1429–1437.

[44] Yuan Yin and Zhewei Wei. 2019. Scalable graph embeddings via sparse transpose proximities. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1429–1437.

[45] Wenchao Yu, Wei Cheng, Charu C Aggarwal, Kai Zhang, Haifeng Chen, and Wei Wang. 2018. Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2672–2681.

[46] Hongyuan Zha and Horst D Simon. 1999. On updating problems in latent semantic indexing. *SIAM Journal on Scientific Computing* 21, 2 (1999), 782–791.

[47] Hongyang Zhang, Peter Lofgren, and Ashish Goel. 2016. Approximate personalized pagerank on dynamic graphs. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 1315–1324.

[48] Jie Zhang, Yuxiao Dong, Yan Wang, Jie Tang, and Ming Ding. 2019. ProNE: Fast and Scalable Network Representation Learning.. In *IJCAI*, Vol. 19. 4278–4284.

[49] Ziwei Zhang, Peng Cui, Haoyang Li, Xiao Wang, and Wenwu Zhu. 2018. Billion-scale network embedding with iterative random projection. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 787–796.

[50] Ziwei Zhang, Peng Cui, Jian Pei, Xiao Wang, and Wenwu Zhu. 2018. Timers: Error-bounded svd restart on dynamic networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[51] Ziwei Zhang, Peng Cui, Xiao Wang, Jian Pei, Xuanrong Yao, and Wenwu Zhu. 2018. Arbitrary-order proximity preserved network embedding. In *Proceedings*

of the 24th ACM SIGKDD international conference on knowledge discovery & data mining. 2778–2786.

[52] Yanping Zheng, Hanzhi Wang, Zhewei Wei, Jiajun Liu, and Sibo Wang. 2022. Instant Graph Neural Networks for Dynamic Graphs. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22)*. 2605–2615.

[53] 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.

[54] 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.

APPENDIX

A INITIALIZATION OF DAMF

Algorithm 3: Initialization of DAMF

Input: A graph $\mathcal{G}$ with adjacency matrix $\mathbf{A}$, embedding dimension d, PageRank damping factor α, error tolerance ϵ.

Output: $\mathbf{X}_b, \mathbf{Y}_b, \mathbf{Z}_b, \mathbf{P}_\mathbf{X}, \mathbf{P}_\mathbf{Y}, \mathbf{r}$

$\mathbf{U}, \mathbf{\Sigma}, \mathbf{V} \leftarrow$ **t-SVD**$(\mathbf{A}, d)$;

$\mathbf{X}_b \leftarrow \mathbf{U}\mathbf{\Sigma}^{1/2}, \quad \mathbf{Y}_b \leftarrow \mathbf{V}\mathbf{\Sigma}^{1/2}$;

$\mathbf{P}_\mathbf{X} \leftarrow \mathbf{I}, \quad \mathbf{P}_\mathbf{Y} \leftarrow \mathbf{I}$;

$\mathbf{r} \leftarrow \mathbf{X}_b, \quad \mathbf{Z}_b \leftarrow \mathbf{O}$;

$\mathbf{Z}_b, \mathbf{r} \leftarrow$ `Propagation`$(\mathbf{Z}_b, \mathbf{r}, \mathcal{G}, \alpha, \epsilon)$;

return $\mathbf{X}_b, \mathbf{Y}_b, \mathbf{Z}_b, \mathbf{P}_\mathbf{X}, \mathbf{P}_\mathbf{Y}, \mathbf{r}$;

Algorithm 3 gives a detailed pseudo-code for the initialization of the DAMF. The random projection-based truncated SVD algorithm [13] is able to complete the t-SVD in $O(nd^2 + md)$ time, while the time complexity of the PPR enhancement that follows the initialization is $O(\frac{ncd}{\alpha\epsilon})$ [52]. Overall, the time complexity of the initialization step is $O(nd^2 + md + \frac{ncd}{\alpha\epsilon})$.

B PROOF

B.1 Proof of Lemma 1

Proof of Lemma 1. The i-th row of the result matrix of the matrix multiplication of $\mathbf{BC}$ can be considered as the i-th row of $\mathbf{B}$ multiplied by the matrix $\mathbf{C}$. Therefore, if the i-th row of $\mathbf{B}$ is all-zero, the i-th row of the result matrix will also be all-zero. Since $\mathbf{B}$ has only t non-zero rows, $\mathbf{BC}$ has at most t non-zero rows. □

B.2 Proof of Proposition 1

Proof of Proposition 1.

$$\begin{aligned}
\left\|(\mathbf{I}-\mathbf{U}\mathbf{U}^\top)\vec{x}\right\|_2 &= \sqrt{((\mathbf{I}-\mathbf{U}\mathbf{U}^\top)\vec{x})^\top(\mathbf{I}-\mathbf{U}\mathbf{U}^\top)\vec{x}} \\
&= \sqrt{\vec{x}^\top(\mathbf{I}-\mathbf{U}\mathbf{U}^\top)^\top(\mathbf{I}-\mathbf{U}\mathbf{U}^\top)\vec{x}} \\
&= \sqrt{\vec{x}^\top(\mathbf{I}-2\mathbf{U}\mathbf{U}^\top+\mathbf{U}\mathbf{U}^\top\mathbf{U}\mathbf{U}^\top)\vec{x}} \\
&= \sqrt{\vec{x}^\top(\mathbf{I}-\mathbf{U}\mathbf{U}^\top)\vec{x}} \\
&= \sqrt{\vec{x}^\top\vec{x}-\vec{x}^\top\mathbf{U}\mathbf{U}^\top\vec{x}} \\
&= \sqrt{\|\vec{x}\|_2^2-\|\mathbf{U}^\top\vec{x}\|_2^2}
\end{aligned} \tag{35}$$

□

B.3 Proof of Lemma 2

Proof. When performing the matrix multiplication of $\mathbf{B}^\top\mathbf{A}$, the element on the i-th row and j-th column of the result matrix can be obtained by the product of the i-th row of $\mathbf{B}^\top$ and the j-th row of $\mathbf{A}$ with

$$(\mathbf{B}^\top\mathbf{A})[i,j] = \mathbf{B}^\top[i]\cdot\mathbf{A}[:,j] \tag{36}$$

Because $\mathbf{B}$ has at most t non-zero rows, $\mathbf{B}^\top[i]$ has at most t non-zero elements. By skipping zeros in calculating the product, the above equation can be calculated with the time complexity of $O(t)$. And since $\mathbf{B}^\top\mathbf{A}$ is a q-by-p matrix, the time complexity of calculating $\mathbf{B}^\top\mathbf{A}$ is $O(tpq)$.

Algorithm 4 is a pseudo-code for the above scheme.

□

Algorithm 4: Algorithm for Lemma 2

Input: $\mathbf{A} \in \mathbb{R}^{n\times p}$, $\mathbf{B} \in \mathbb{R}^{n\times q}$ and $\mathbf{B}$ has t non-zero rows.

Output: $\mathbf{B}^\top\mathbf{A}$

$\mathbf{C} \leftarrow \mathbf{O}_{q\times p}$;
foreach *l with $\mathbf{B}[l]$ is non-zero* **do**
  for $i \leftarrow 1$ *to* q **do**
    for $j \leftarrow 1$ *to* p **do**
      $\mathbf{C}[i,j] \leftarrow \mathbf{C}[i,j] + \mathbf{A}[l,j]\times\mathbf{B}[l,i]$;
    end
  end
end
return $\mathbf{C}$

B.4 Proof of Lemma 3

Proof. From Lemma 1, there are only at most t non-zero rows in the result of $\mathbf{BC}$. So, by skipping the calculation of all-zero rows, the time complexity of of calculating $\mathbf{BC}$ is $O(tpq)$.

Algorithm 5 is a pseudo-code for the above scheme. □

Algorithm 5: Algorithm for Lemma 3

Input: $\mathbf{B} \in \mathbb{R}^{n\times q}$ and $\mathbf{B}$ has t non-zero rows, $\mathbf{C} \in \mathbb{R}^{q\times p}$

Output: $\mathbf{BC}$

$\mathbf{D} \leftarrow \mathbf{O}_{n\times p}$;
foreach *l with $\mathbf{B}[l]$ is non-zero* **do**
  for $i \leftarrow 1$ *to* q **do**
    for $j \leftarrow 1$ *to* p **do**
      $\mathbf{D}[l,j] \leftarrow \mathbf{D}[l,j] + \mathbf{B}[l,i]\times\mathbf{C}[i,j]$;
    end
  end
end
return $\mathbf{D}$

C DETAILED EXPERIMENTAL SETTINGS

All experiments are conducted using 8 threads on a Linux machine powered by an AMD Epyc 7H12@3.2GHz and 768GB RAM. For baselines that require a GPU, we used an additional NVIDIA GeForce RTX 3090 with 24G memory. The experimental results for each task are the average of the results of five experiments.

C.1 Additional Details of Link Prediction

The objective of link prediction is, for the directed node pair (u, v), to predict whether there is a directed edge from u to v. For each pair of nodes (u, v) in the test set, we determine a score by taking the inner product of u's context vector and v's content vector. For methods that do not distinguish between context embedding and content embedding, we set both their context embedding and content embedding to be node embeddings. For undirected graphs, we calculate link prediction scores for both directions separately, and then select the larger value as the score.

C.2 Additional Details of Graph Reconstruction

For *Wiki* and *Cora* datasets, we define set $\mathcal{S}$ as the collection of all conceivable node pairs. On *Flickr*, we construct $\mathcal{S}$ by taking a 1% sample of all possible pairs of nodes. Figure 4 depicts the performance of all methods in the graph reconstruction task with K values ranging from 10^1 to 10^6. DAMF performs better than its competitors on all the datasets and for nearly every K value. The remarkable accuracy of DAMF is especially noticeable on the *Wiki* and *Cora* datasets as K increases.

C.3 Dataset

Wiki [22] is a hyperlinked network of Wikipedia. Each node in the directed graph represents a page, and the edges represent hyperlinks.

Cora [15] is a dynamic citation undirected network where each node represents a paper, the edges represent citations, and each article is labeled with the domain to which the article belongs.

Flickr [36] is an undirected network of user links on Flickr, where nodes represent users and labels are groups of interest to the user.

YouTube [37] is a video-sharing website where users can upload and watch videos, and share, comment and rate them. Each user is labeled with their favorite video genre.

Orkut [23] is an undirected social network where nodes represent users and edges represent user interactions. Users are labeled with the community they are in.

Twitter [17] is a massive-scale directed social network where nodes represent users and edges represent the following relationship.

C.4 Code

We use the following code as our baseline:

Dynnode2vec [21]: https://github.com/pedugnat/dynnode2vec
GloDyNE [15]: https://github.com/houchengbin/GloDyNE
DynGEM [11]: https://github.com/palash1992/DynamicGEM
DNE [8]: https://github.com/lundu28/DynamicNetworkEmbedding
RandNE [49]: https://github.com/ZW-ZHANG/RandNE

As we could not find the code for LocalAction [19], we re-implemented the pseudo-code from the paper in Python and made our implementation available along with our experimental code.

Our code for experiments is available on:
https://github.com/zjunet/DAMF