Efficient Graph Embedding Generation and Update for Large-Scale Temporal Graph

Yifan Song
HKUST(GZ)
ysong853@connect.hkust-gz.edu.cn

Xiaolong Chen
HKUST(GZ)
xchen738@connect.hkust-gz.edu.cn

Wenqing Lin
Tencent
edwlin@me.com

Jia Li
HKUST(GZ) & HKUST
xiyou3368@gmail.com

Chen Zhang
Zhejiang CreateLink Technology
zhangchen@chuanglintech.com

Yan Zhou
Zhejiang CreateLink Technology
zhouyan@chuanglintech.com

Lei Chen
HKUST(GZ) & HKUST
leichen@cse.ust.hk

Jing Tang
HKUST(GZ) & HKUST
jingtang@ust.hk

ABSTRACT

Graph embedding aims at mapping each node to a low-dimensional vector, beneficial for various applications like pattern matching, retrieval augmented generation and recommendation. In this paper, we study the large-scale temporal graph embedding problem. Different from simple graphs, each edge has a timestamp in temporal graphs, which requires the embeddings to encode the temporal biases. Factorizing similarity matrix is a common approach for generating simple graph embeddings where similarity can be well characterized by some conventional metrics like personalized PageRank. However, how to construct a similarity that can encode interactions with temporal biases is a critical problem for large scale temporal graphs. To address this, we introduce the concept of temporal-based bipartite graph (TBG) and develop the temporal preferential attachment similarity (TPASim) that reflects concurrent node activity over time. Directly factorizing the TPASim matrix, which contains nearly n^2 non-zeros, is not feasible for large graphs with n nodes. Instead, we present LTGE, which constructs and factorizes a temporal matrix with at most $2m$ non-zeros, where m is the number of edges. Our theoretical analysis shows that LTGE achieves the same embeddings as factorizing the TPASim matrix but significantly reduces complexity by a factor of $\frac{n^2}{m}$. On the other hand, when graphs evolve over time, to avoid recomputing, we further propose LTGEInc that utilizes a novel incremental singular value decomposition (SVD) algorithm with provable guarantee for updating the embeddings. Extensive experiments on several datasets with up to 17 million nodes and 1.3 billion edges demonstrate that LTGE outperforms the state of the art significantly and is orders of magnitude faster than the baselines specially designed for temporal graphs. For embeddings update, LTGEInc retains the performance with small computational overhead.

PVLDB Reference Format:
Yifan Song, Xiaolong Chen, Wenqing Lin, Jia Li, Chen Zhang, Yan Zhou, Lei Chen, and Jing Tang. Efficient Graph Embedding Generation and Update for Large-Scale Temporal Graph. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://anonymous.4open.science/r/VLDB2025LTGE-DE38/.

1 INTRODUCTION

Graph is a common data structure in real world which can model complex relationships. For instance, it can represent associations between users and products on shopping platforms or interactions in biomolecular systems. For graph analytics, graph embedding is a fundamental task aiming to represent graph as low-dimensional embeddings for downstream tasks such as link prediction [50, 61, 62] and recommendation [7, 58, 64]. There are a plethora of techniques for generating graph embeddings, with early approaches utilizing random walks on the graph as sentences to train word2vec models [18, 46]. Later, Liu et al. [40] provide a general view for using matrix factorization (MF) to obtain graph embeddings with thorough theoretical analysis. Since then, several MF-based approaches [15, 48, 61, 62, 67] are proposed to improve the efficiency of generating graph embeddings. Nevertheless, the aforementioned methods mainly focus on simple graphs, without considering the temporal information in temporal graphs.

In recent years, some work studies the temporal graph embedding problem [9, 37, 51, 70]. In a temporal graph, each edge is associated with a timestamp that represents when the edge is added, which introduces the temporal biases. For example, Figure 1(b) is a temporal co-author graph collected from 2022 to 2023, where every edge has a timestamp about their generation time, while Figure 1(a) is the simple co-author graph at 2023 with the same structure. Considering temporal interactions, node 9 has a more recent publication co-authored with node 7 than node 6, indicating a closer collaboration with the former. In the view of simple graph at 2023 by ignoring the timestamps, with respect to node 7, node 6 and node 9 are structurally equivalent. As a result, existing temporal graph embedding methods aim to capture the temporal biases in temporal graphs.

However, previous temporal graph embedding methods are hard to scale to large graphs with billions of edges due to prohibitive training time and/or memory usage. To address the deficiency of previous work, we propose LTGE via temporal similarity matrix factorization tailored to **L**arge **T**emporal **G**raph **E**mbedding. The challenge is two-fold, which requires (i) a proper temporal similarity metric and (ii) an efficient factorization. For simple graphs, the similarity between two nodes can be well characterized by some conventional metrics such as personalized PageRank [28], preferential attachment similarity [2], and SimRank [27]. Such metrics, unfortunately, cannot capture temporal information.

Inspired by the preferential attachment similarity [2], we divide time into a sequence of snapshots, and create an edge between a node and a snapshot if the node is active in the snapshot (i.e., the node interacts with some other nodes in the snapshot). Moreover, if a node has more interactions in one snapshot, the weight of the corresponding edge is larger. In this way, we create a temporal bipartite graph (TBG) consisting of the original nodes, snapshots, and node-snapshot edges. Then we propose a new similarity measure based on TBG, called temporal preferential attachment similarity (TPASim). That is, the TPASim between two nodes is the logarithm weight products of them appearing in the same snapshot. We show that if ignoring temporal information, i.e., simple graphs, TPASim is exactly a preferential-attachment-like similarity. Furthermore, factoring a similarity matrix with n^2 elements is computationally expensive, where n is the number of nodes in the graph. According to the definition of TPASim, we prove that the singular value decomposition (SVD) of the similarity matrix can be easily obtained from the SVD of a smaller matrix with $n \times k$ dimensions, among which $O(m)$ are non-zeros, where m is the number of edges in the graph and k is the number of snapshots. As a result, the efficiency is significantly improved.

On the other hand, as real-world graphs always evolve over time, the embeddings shall be updated accordingly. Considering the example given in Figure 1, for the embeddings obtained at 2022, it is obvious that node 8, node 9 and node 10 are unseen unless we update their embeddings at 2023. Previous studies on embedding update [13, 22, 38] focus on simple dynamic graphs without considering temporal interactions. Therefore, another challenge for temporal graph embedding is to deal with evolving nodes and edges while reserving temporal information. A naive solution is to recompute whenever needed, which is computationally expensive and time-consuming. In this paper, we propose LTGEInc that can update the embeddings incrementally while retaining temporal interactions. Basically, LTGEInc is based on a novel incremental SVD that applies to our temporal similarity matrix. Moreover, through a rigorous analysis, we show that the error introduced by LTGEInc is theoretically bounded. It supports LTGEInc to perform well in downstream tasks even with multiple updates to the embeddings without significant degradation under limited error accumulation.

Finally, the effectiveness and efficiency of our proposed methods are demonstrated through experiments on multiple datasets. In particular, LTGE outperforms all competitors on future link prediction and top-K recommendation, while LTGEInc shows its superior capability on incremental future link prediction task. Moreover, previous temporal graph embedding baselines are hard to derive embeddings on million-scale graphs, whereas our LTGE finishes within 9 hours on the largest dataset tested, i.e., Taobao with 17.9 million nodes and 1.3 billion edges.

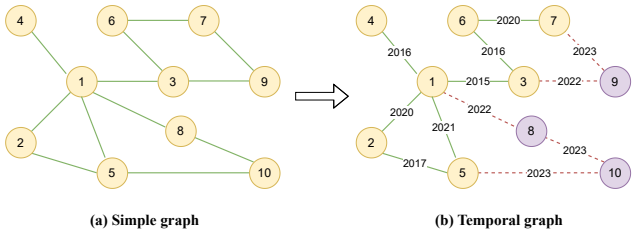


Figure 1: Simple graph vs. temporal graph.

In summary, our contributions in this paper are as follows.

- We propose LTGE via temporal similarity matrix factorization for large temporal graph embedding. In particular, we devise a novel similarity measure TPASim that can encode temporal interactions effectively and can be factorized in an efficient way.
- We further develop LTGEInc that updates embeddings incrementally with provable guarantees as graph evolves, instead of intractable recomputing.
- We evaluate LTGE on ten public datasets against eleven competitors and LTGEInc on three large dynamic graphs against three methods with high efficiency, respectively. The experimental results demonstrate the superiority of LTGE and LTGEInc. In particular, for the Taobao dataset with 17.9 million nodes and 1.3 billion edges, all the temporal graph embedding methods fail to produce embeddings except LTGE. Meanwhile, LTGEInc outperforms all competitors in the incremental task with the highest efficiency.

Roadmap. In Section 2, we review the related work on the graph embedding problem. Problem formulation is given in Section 3. To address the problem, in Sections 4 and 5, we present LTGE and its incremental update version LTGEInc, respectively. Section 6 conducts complexity analysis. Performance of our solutions is evaluated in Section 7. Finally, we conclude our paper in Section 8.

2 RELATED WORK

Simple Graph Embedding. Techniques such as locally linear embedding [52] and Laplacian eigenmaps [3] are significant early contributions. However, these methods can only capture the local structure to generate low-quality embeddings. Later, the advent of word embedding models inspires researchers for generating graph embedding. DeepWalk [46] considers random walks as sentences, and samples several random walks on the graph to train skip-gram model [41] to get the graph embeddings. On the basis of DeepWalk, node2vec [18] adjusts two parameters to control the random walk, thus obtaining more structural information and providing more adaptable embeddings. LINE [55] defines the first-order and second-order similarity between nodes and optimizes the skip-gram model [41] through negative sampling. HuGE [15] utilizes a hybrid-property heuristic random walk to capture structural information and uses a heuristic algorithm to determine parameters of the random walk. Some methods [1, 8, 10, 57, 68] apply deep learning to graph embeddings, but they tend to be computationally expensive on large graphs.

To further improve the efficiency and provide comprehensive theoretical analysis, many efficient matrix-factorization-based approaches [47, 48, 61, 62, 67] are proposed for generating graph embeddings. Specifically, NRP [61] designs node-reweighting pagerank, which is an effective similarity measure for generating embeddings efficiently. NetSMF [48] utilizes spectral sparsification theories to improve the efficiency of matrix factorization. ProNE [67] enhances the embeddings by disseminating them within the space modulated spectrally on the basis of sparse matrix factorization. However, the aforementioned methods overlook the temporal information in temporal graphs. To address this issue, temporal graph embedding has garnered increased attention in recent years.

Temporal Graph Embedding. Temporal graph embedding methods aim to utilize temporal information to enhance the quality of embeddings. For temporal graph embedding, CTDNE [43] introduces the temporal random walk to encode temporal information. EHNA [25] later improves upon this temporal random walk and designs a deep neural network to learn high-quality embeddings. Over recent years, temporal graph neural network (T-GNN) has emerged as potent models for learning representations on temporal graphs. TGAT [9] utilizes the message-passing scheme of simple GNNs and incorporates time information to obtain temporal graph embeddings. TGN [51] leverages a memory module to track historical interactions, thereby gaining temporal information and achieving significant accuracy improvements over simple graph embedding methods. APAN [59] expedites model inference via asynchronous message propagation. Furthermore, the state-of-the-art T-GNN model Zebra [37] employs a top-k T-PPR to reduce neural network layers, demonstrating high effectiveness in future link prediction task.

However, even for the most efficient temporal graph embedding method Zebra, the memory module update remains time-consuming, and storing historical interactions requires substantial memory, restricting its applicability for massive graphs. Meanwhile, T-GNNs need node feature to train the model, which is not always available for different downstream tasks [6, 13]. Also, all of the aforementioned T-GNN models cannot update the embeddings for new nodes using the model trained on the original subgraph. Nevertheless, practical applications often demand incremental tasks, which require that the embeddings can be adjusted with new data instead of generating embedding based on subgraph knowledge [13, 54].

Embedding Update for Dynamic Graphs. There are several studies that aim at efficiently updating the embeddings with the graph structure changes, which are also called dynamic graph embedding. DNE [12] modifies the embeddings of nodes by tuning the skip-gram model to accommodate the dynamic nature of graphs. LocalAction [39] randomly samples the neighbors of the updated node and modifies the embeddings of the sampled node. This method is highly efficient, but it performs poorly in practical tests because it does not consider high-order proximity. DynGEM [17] uses an autoencoder to obtain graph embeddings and avoids retraining by means of regularization parameter passing. GloDyNE [22] designs a novel node selection strategy combined with a novel incremental learning paradigm to enhance the capture of high-order proximity changes. DAMF [11] is the state-of-the-art dynamic graph embedding method for large dynamic graph, which updates the embeddings via space projection and uses personlized pagerank to enhance its embeddings. However, most dynamic graph embedding methods only focus on how to efficiently update the embedding when the graph structure changes and ignore the temporal information brought by the timestamp, so they always perform worse than temporal graph embedding methods on temporal graphs.

Table 1: Frequently used notations.

Notation	Description
$G = (V, \mathcal{E})$	A temporal graph G with node set V, edge set $\mathcal{E}$, where each $(v_i, v_j) \in \mathcal{E}$ is associated with a timestamp t_{v_i,v_j}
n, m	The numbers of nodes and edges in G
d	The dimension of embedding
k	The number of snapshots
U_T	The temporal node set generated by snapshots
G_t=$(V_t, \mathcal{E}_t)$	A bipartite graph G_t with node set V_t=$(V \cup U_T)$ and edge set $\mathcal{E}_t \subseteq V \times U_T$
$\mathrm{w}(v_i, u_j)$	The weight of edge in G_t between nodes $v_i \in V$ and $u_j \in U_T$
$\mathrm{TPASim}(v_i, v_j)$	Temporal preferential attachment similarity between nodes v_i and v_j in temporal-based bipartite graph G_t
$\mathbf{W} \in \mathbb{R}^{n \times k}$	The temporal matrix of G_t such that $\mathbf{W}[i, j] = \ln \mathrm{w}(v_i, u_j)$

In this paper, we (i) develop scalable temporal graph embedding algorithms for large graphs with millions of nodes, and (ii) propose an efficient method for update embeddings incrementally when graph evolves.

3 PROBLEM FORMULATION

3.1 Preliminaries

In the context of directed temporal graphs, a temporal graph denotes as $G = (V, \mathcal{E})$, where V represents the set of nodes, and $\mathcal{E}$ represents the set of edges, and each edge $(v_i, v_j) \in \mathcal{E}$ is annotated with a timestamp t_{v_i,v_j} indicating the generation timestamp of the edge. Temporal graph embedding aims to project each node $v_i \in V$ into a low-dimensional vector space, represented by $\mathbf{z}_{v_i}$, that encapsulates both the topological structure and the temporal features of the node. High quality embeddings are expected to facilitate accurate predictions in various tasks, such as future link prediction [25, 43] and top-K recommendations [30]. Our work firstly studies the problem of generating high quality temporal graph embeddings. Also, as real-world graphs always evolve over time, this work further delves into strategies for incrementally updating temporal graph embeddings, with a particular focus on the addition of edges and nodes, which is crucial for many important applications. For example, in citation networks, new citations (i.e., edges) and new papers (i.e., nodes) are produced over time, while in e-commerce networks, there are new purchases (i.e., edges) and new customers (i.e., nodes) everyday.

Notations. For a given temporal graph $G = (V, \mathcal{E})$, let $n = |V|$ and $m = |\mathcal{E}|$ be the number of nodes and edges in G, respectively. Denote by d the dimension of embeddings, where $d \ll n$. All matrices in this

paper will use bold uppercase. In particular, $\mathbf{Z} \in \mathbb{R}^{n\times d}$ represents the graph embeddings, while $\mathbf{z}_i$ denotes the i-th row vector. G_t is a temporal-based bipartite graph with node set $V_t = (V \cup U_T)$ and edge set $\mathcal{E}_t \subseteq V \times U_T$, where U_T is the node set generated by snapshots. We denote $\mathrm{w}(v_i, u_j)$ as the edge weight between node v_i and u_j in G_t, while $\mathbf{W} \in \mathbb{R}^{n\times k}$ is the temporal matrix of G_t such that $\mathbf{W}[i, j] = \ln \mathrm{w}(v_i, u_j)$. Moreover, $\|\mathbf{X}\|_F$ means the Frobenius norm of $\mathbf{X}$, while $\sigma_i(\mathbf{X})$ means the i-th singular value of $\mathbf{X}$. Table 1 shows the frequently used notations in this paper.

3.2 Matrix Factorization for Graph Embedding

To generate graph embeddings from the similarity matrix, many previous studies [48, 49, 60, 61] use SVD to get a primary high-dimension representation $\mathbf{H} \in \mathbb{R}^{n\times n}$ and then set the graph embedding $\mathbf{Z} = \mathbf{H}_d \in \mathbb{R}^{n\times d}$ as the top-d columns of $\mathbf{H}$ to reduce the dimensions. That is, given a similarity matrix $\mathbf{S} \in \mathbb{R}^{n\times n}$, SVD derives that

$$\begin{aligned} \rho\mathbf{S} &= \mathbf{U}\Sigma\mathbf{V}^\top, \\ \text{and} \quad \mathbf{H} &= \mathbf{U}\sqrt{\Sigma}, \end{aligned} \tag{1}$$

where $\mathbf{U}$ and $\mathbf{V}$ are $n \times n$ complex unitary matrices, Σ is an $n \times n$ rectangular diagonal matrix with non-negative real numbers on the diagonal, and ρ is a parameter for adjusting the value of embeddings. In fact, leveraging the framework developed by Liu et al. [40], $\mathbf{H}$ is the solution of maximizing the following objective function with respect to vector $y_i \in \mathbb{R}^{1\times n}$ for each node v_i.

$$O(v_i, v_j) = \sigma_\rho(s_{i,j}) \ln \sigma(y_i y_j^\top) + \sigma_\rho(-s_{i,j}) \ln \sigma(-y_i y_j^\top), \tag{2}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function, $\sigma_\rho(x) = \frac{1}{1+e^{-\rho x}}$, and $s_{i,j}$ is the similarity between nodes v_i and v_j. Here, this objective function follows the target that two similar (resp. dissimilar) nodes v_i and v_j in a graph, i.e., large (resp. small) value of $s_{i,j}$, are also close (resp. far away) in embedding space, i.e., large (resp. small) value of $y_{i,j} = y_i y_j^\top$.

To maximize (2), one can treat $y_{i,j}$ as an independent value [36]. Considering the local objective function for a node pair (v_i, v_j), we can get the partial derivative with respect to $y_{i,j}$ that

$$\begin{aligned} \frac{\partial O(v_i, v_j)}{\partial y_{i,j}} &= \sigma_\rho(s_{i,j})\big(1 - \sigma(y_{i,j})\big) - \sigma_\rho(-s_{i,j})\big(1 - \sigma(-y_{i,j})\big) \\ &= \sigma_\rho(s_{i,j})\big(1 - \sigma(y_{i,j})\big) - \big(1 - \sigma_\rho(s_{i,j})\big)\sigma(y_{i,j}), \end{aligned}$$

where the first equality is due to $\frac{\partial \sigma(x)}{\partial x} = \sigma(x)\big(1 - \sigma(x)\big)$ and the second equality is due to $\sigma(-x) = 1 - \sigma(x)$ and $\sigma_\rho(-x) = 1 - \sigma_\rho(x)$. Setting $\frac{\partial O(v_i, v_j)}{\partial y_{i,j}} = 0$, we can get that

$$y_{i,j} = \rho \cdot s_{i,j}.$$

Alternatively, letting y_i be the i-th row of matrix $\mathbf{Y}$, we can write in the form of matrix that

$$\mathbf{Y}\mathbf{Y}^\top = \rho\mathbf{S}, \tag{3}$$

where $s_{i,j}$ is the i-th row and j-th column element of the similarity matrix $\mathbf{S} \in \mathbb{R}^{n\times n}$. Note that when $\mathbf{S}$ is a semi-positive definite matrix we have $\mathbf{U} = \mathbf{V}$ [16]. According to (1), it is trivial to see that $\mathbf{H}\mathbf{H}^\top = \rho\mathbf{S}$. This implies that $\mathbf{Y} = \mathbf{H}$ is the solution to maximize (2).

At the same time, truncated SVD of $\mathbf{S}$ is the best d-rank approximation of $\mathbf{S}$ in Frobenius space, which means $\mathbf{Z} = \mathbf{H}_d$ is the solution of the following equation.

$$\min_{\mathbf{Z}\in\mathbb{R}^{n\times d}} \|\mathbf{Z}\mathbf{Z}^\top - \mathbf{H}\mathbf{H}^\top\|_F, \tag{4}$$

Thus $\mathbf{Z}$ is a good d-dimension representation for $\mathbf{H}$. Equations (1)–(4) aim to make the embeddings retain the similarity information on the graph. The above analysis indicates that for a well-defined similarity measure, truncated SVD on the similarity matrix is suitable for generating graph embeddings. On this basis, the primary problem of temporal graph embedding is how to define a new similarity measure that can encode the temporal information. Moreover, we find that it is also important for temporal graph embedding methods to have the ability to update rapidly facing addition of nodes or edges for practical application. In the following content, we state how to solve the issues mentioned above through a well-defined similarity TPASim and an incremental SVD algorithm to handle the embeddings update problem.

4 THE LTGE ALGORITHM

4.1 Temporal Distribution Similarity

Given a temporal graph $G = (V, \mathcal{E})$, we firstly define a snapshot as a subset of nodes and temporal edges within a time range. Specifically, for a temporal graph with time span T, each snapshot contains nodes and edges within a fixed length of time $\tilde{T}_i$ and $\sum_i \tilde{T}_i = T$. One kind of intuitive temporal information is that nodes that appear in the same snapshot may have similar temporal bias in behavior. Based on the idea, we first divide the temporal graph into k snapshots, while each snapshot has the $\frac{T}{k}$ length of time.

After splitting the graph, the entire edge set is divided into k different subsets across distinct time intervals with each snapshot. These subsets are denoted as $E_1, E_2, ..., E_k$, where the intersection of each subset is empty (i.e., $\forall_{i\neq j}, E_i \cap E_j = \emptyset$) and the union of these sets forms the entire edge set $\mathcal{E}$ (i.e., $\bigcup_i^k E_i = \mathcal{E}$). We treat each of the k subsets as new nodes, labeled as $u_1, u_2, ..., u_k \in U_T$, where u_l corresponds to E_l. Each $u_i \in U_T$ denotes a temporal node, while U_T is the temporal node set. Every temporal node represents a period of time, thus we can build the connection between temporal nodes and original nodes to model the temporal information preference of each node.

Then we initialize a new graph $G_t = (V_t = (V \cup U_T), \mathcal{E}_t = \emptyset)$ which contains both the original nodes and the temporal nodes. Next, we generate the edges that associate these two types of nodes. For each $\ell = 1, 2, ..., k$, we perform the following operations on each edge $e = (v_i, v_j)$ in $\mathcal{E}$:

(1) If there is no edge (v_i, u_ℓ) in the edge set $\mathcal{E}_t$ of the new graph, then add an edge (v_i, u_ℓ) with weight t_{v_i,v_j}; if an edge (v_i, u_ℓ) already exists, then add the weight of the edge by t_{v_i,v_j}.
(2) Perform the same for the edge (v_j, u_ℓ).

After above operations, we construct a bipartite graph that can explicitly represent the connection between nodes and different time intervals, which is called **T**emporal-based **B**ipartite **G**raph (**TBG**) in the following content.

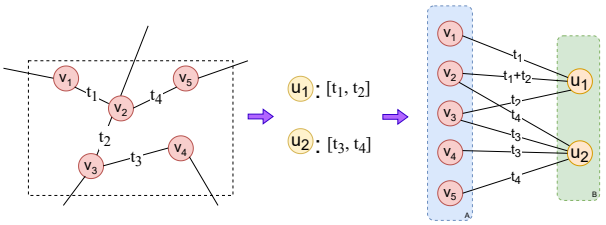


Figure 2: Example of temporal bipartite graph conversion (the left picture is the original temporal graph G, the center is the generated temporal node, and the right picture is the converted bipartite graph G').

Example. We provide an example of how to generate a TBG from a general temporal graph. In the left of Figure 2, there is a subgraph G with five nodes and four edges. As stated above, if we set $k = 2$ for G, all edges in G can be divided into two snapshots $E_1 = \{(v_1, v_2), (v_2, v_3)\}$ and $E_2 = \{(v_3, v_4), (v_2, v_5)\}$ based on their generation time. For $E_1 = \{(v_1, v_2), (v_2, v_3)\}$, we create a new node u_1 to represent the snapshot and generate new edges (v_1, u_1), (v_2, u_1), and (v_3, u_1), where u_1 is called a temporal node. Since nodes v_1 and v_3 each appear in an edge of E_1, as stated above, edge (v_1, u_1) has a weight t_1 and (v_3, u_1) has a weight of t_2, while node v_2 appears in both edges and hence the edge (v_2, u_1) has a weight of $t_1 + t_2$. Similarly, we can generate edges for temporal node u_2. Finally, the TBG is constructed with original node set A and temporal node set B. The temporal-based bipartite graph after the construction is shown in the right of the figure.

Temporal-based bipartite graph could effectively retain the time sequence information of the original temporal graph by connecting the original graph nodes to the corresponding temporal nodes. Next task is to find a similarity measurement on TBG to capture the structure and temporal information simultaneously. Previous studies like Personalized PageRank [28] or SimRank [27] just focus on the connection between original nodes, which only consider structure information but ignore the temporal information in temporal graph.

Thus, we propose a new similarity called **TPASim**, that concentrates on temporal distribution around each node. The motivation behind is that two nodes are close when they appear in same temporal snapshots frequently, because it means their activity trajectory is coincident partly to some extent. In the constructed TBG, the similarity between two original nodes is determined by the edges they connect to the same temporal nodes, which is defined as follow:

Definition 4.1. Given a TBG $G_t = (V_t, \mathcal{E}_t)$, the temporal preferential attachment similarity (TPASim) between any pair of graph nodes v_i and v_j is

$$\mathrm{TPASim}(v_i, v_j) = \sum_{u_\ell \in U} \ln \mathrm{w}(v_i, u_\ell) \cdot \ln \mathrm{w}(v_j, u_\ell).$$

Here $\mathrm{w}(v_i, u_l)$ represents the weight of the edge between node v_i and node u_l. The similarity definition takes the time distribution related to nodes into account, where two nodes are more similar when the generation time distribution of the edges around two nodes are closer.

In fact, TPASim is inspired by preferential attachment similarity [2] that is a common measurement in simple graph defined as $\mathrm{PASim}(v_i, v_j) = d_i \cdot d_j$. Preferential attachment similarity has been used in many areas such as percolation [21], transportation [66] and synchronization [63]. The following lemma shows the connection between TPASim and preferential attachment similarity.

Lemma 4.2. *For a simple graph, which means the timestamp t for every edge is a constant $t_0 > 1$ and the number of snapshots $k = 1$ is the only reasonable choice, we have*

$$\mathrm{TPASim}(v_i, v_j) = \ln d_i \ln d_j + \ln t_0 \ln \mathrm{PASim}(v_i, v_j) + \ln^2 t_0.$$

Proof. Because $k = 1$, there is only one temporal node in TBG, which means that $\forall v \in V, \mathrm{w}(v, 1) = \ln(\Sigma_1^{d_v} t_0) = \ln(t_0 \cdot d_v)$, where d_v is degree of node v. So we can get:

$$\begin{aligned}
\mathrm{TPASim}(v_i, v_j) &= \mathrm{w}(v_i, 1) \cdot \mathrm{w}(v_j, 1) \\
&= (\ln(\Sigma_1^{d_{v_i}} t_0)) \cdot (\ln(\Sigma_1^{d_{v_j}} t_0)) \\
&= (\ln(t_0) + \ln d_{v_i}) \cdot (\ln(t_0) + \ln d_{v_j}) \\
&= \ln d_i \ln d_j + \ln t_0 \ln \mathrm{PASim}(v_i, v_j) + \ln^2 t_0,
\end{aligned}$$

which completes the proof. □

In conclusion, with constructing TBG and computing TPASim, we could simultaneously consider structure information and temporal information. In what follows, we show that the embeddings can be generated efficiently through factorizing a small matrix, which avoids computing and factorizing the whole TPASim matrix.

4.2 Efficient Matrix Factorization for Temporal Graph Embedding

As stated in Section 3.2, we should factorize the similarity matrix to generate the final embeddings [40]. However, it is inefficient to factorize the whole similarity matrix, thus we next give analysis on how to get the same result by factorizing a small matrix. First, we define the **temporal matrix**, which represents the logarithm of weighted connection relationship between the original nodes and the temporal nodes on TBG, as follow:

$$\mathbf{W}[i, j] = \ln \mathrm{w}(v_i, u_j),$$

where $v_i \in V$ is an original node, $u_j \in U_T$ is a temporal node and $\mathbf{W} \in \mathbb{R}^{n \times k}$ is the temporal matrix where $\mathbf{S} = \mathbf{W}\mathbf{W}^\top$. Note that for each edge (v_i, v_j) in the original temporal graph G, at most two edges are created in the corresponding TBG G_t that connect v_i and v_j to one temporal node u_ℓ. Hence, G_t has at most $2m$ edges, indicating that $\mathbf{W}$ contains $O(m)$ non-zero elements, where $m \ll n^2$ in general, n and m are the number of nodes and edges in G .

Next, the following lemma can prove that the objective function could be maximized by just factorizing the temporal matrix $\mathbf{W} \in \mathbb{R}^{n \times k}$ with SVD factorization.

Lemma 4.3. *Let $\mathbf{Z} \in \mathbb{R}^{n \times d}$ be matrix obtained by the objective function* (1) *and do the singular value decomposition (SVD) of the temporal matrix $\mathbf{W}$ that $\mathbf{W} = \mathbf{U}\Sigma\mathbf{V}^\top$. We then have:*

$$\mathbf{Z} = \sqrt{\rho}\mathbf{U}_d\Sigma_d,$$

where $\mathbf{U}^\top\mathbf{U} = \mathbf{I}$, $\mathbf{V}^\top\mathbf{V} = \mathbf{I}$, and $\mathbf{I}$ is identity matrix. $\mathbf{U}_d$ and Σ_d represent the top-d columns of each matrix.

Algorithm 1: LTGE

Input: Temporal graph $G = (V, \mathcal{E})$, the whole time span T, the number of snapshots k, embedding dimension d, parameter ρ.
Output: Graph embeddings $\mathbf{Z}$, right singular matrix $\mathbf{V}_d^\top$

$t_{\text{label}} \leftarrow 0$, size $\leftarrow \frac{T}{k}$, snapshot $\leftarrow 0$;
Initialize a null matrix $\mathbf{W} \in \mathbb{R}^{n \times k}$;
for $(v_i, v_j) \in \mathcal{E}$ **do**
  if $t_{v_i,v_j} - t_{\text{label}} >$ size **then**
    $t_{\text{label}} \leftarrow t_{v_i,v_j}$;
    snapshot $\leftarrow$ snapshot + 1;
  $\mathbf{W}[v_i, \text{snapshot}] \leftarrow \mathbf{W}[v_i, \text{snapshot}] + t_{v_i,v_j}$;
  $\mathbf{W}[v_j, \text{snapshot}] \leftarrow \mathbf{W}[v_j, \text{snapshot}] + t_{v_i,v_j}$;
$\mathbf{W} \leftarrow \sqrt{\rho} \ln \mathbf{W}$ for all nonzero elements in $\mathbf{W}$;
$\mathbf{U}_d, \Sigma_d, \mathbf{V}_d^\top \leftarrow \text{RandomizedSVD}(\mathbf{W})$;
$\mathbf{Z} \leftarrow \mathbf{U}_d \Sigma_d$;
return $\mathbf{Z}, \mathbf{V}_d^\top$;

Proof. Let $\mathbf{S}[i, j] = \text{TPASim}(v_i, v_j)$. It has been proved that Equation (4) can be optimized by factorizing the similarity matrix $\mathbf{S}$. According to the definition of TPASim, we have $\mathbf{S} = \mathbf{W}\mathbf{W}^\top$. Meanwhile, $\mathbf{W} = \mathbf{U}\Sigma\mathbf{V}^\top$ and $\mathbf{V}^\top\mathbf{V} = \mathbf{I}$. Thus we can get:

$$\begin{aligned}\mathbf{H}\mathbf{H}^\top &= \rho \cdot \mathbf{S} = \rho \mathbf{W}\mathbf{W}^\top \\ &= \rho \cdot \mathbf{U}\Sigma\mathbf{V}^\top(\mathbf{U}\Sigma\mathbf{V}^\top)^\top \\ &= (\sqrt{\rho} \cdot \mathbf{U}\Sigma)(\sqrt{\rho} \cdot \mathbf{U}\Sigma)^\top.\end{aligned}$$

So that $\mathbf{Z} = \mathbf{H}_d = \sqrt{\rho}\mathbf{U}_d\Sigma_d$. □

To further improve the efficiency of LTGE, we use Randomized SVD [19] to calculate the final embeddings. Its computational time primarily depends on the non-zero elements in the temporal matrix, i.e., $O(m)$. Finally, Algorithm 1 illustrates the pseudo-code of LTGE.

5 AN INCREMENTAL METHOD: LTGEINC

When the initial graph contains only a small number of edges (e.g., newly constructed graph datasets), the final graph may have far more nodes and edges than the original subgraph with newly adding data, and the topological structure of the graph will also be changed. In this case, it is difficult for the embeddings trained on the initial subgraph to obtain the knowledge applicable to the final large-scale graph, which leads to a serious decline in the quality of embeddings.

A naive approach to handle this issue is to retrain the model with fixed time intervals. However, existing temporal graph embedding methods require a significant amount of time to generate the embeddings, while multiple recalculations necessitate extensive time and computing resources. To solve this problem, we propose an incremental algorithm named LTGEInc. It avoids repeatedly recomputing and has a theoretical guarantee.

Example. Figure 3 gives a toy example of the temporal matrix of Figure 2 and how it changes when new edges (v_2, v_5) at time t_5 and (v_5, v_6) with new node v_6 at time t_6 are added. Since the timestamps of new edges are larger than any existing edge's, they will only add new temporal nodes in TBG, which will only append new columns to original temporal matrix. For newly added nodes, they do not have interaction with existing temporal nodes, so that the newly added row will have zeros below the original temporal matrix. Motivated by this property of the temporal matrix, we can use incremental SVD to complete incremental embedding update.

Algorithm 2: IncrementalSVD

Input: Factorization of original matrix $\mathbf{U_1}\Sigma_\mathbf{1}$ and $\mathbf{V}_1^\top$, factorization of new added matrix $\mathbf{U}_2\Sigma_2$ and $\mathbf{V}_2^\top$.
Output: Incremental truncated SVD of new matrix $\mathbf{U}_I\Sigma_I\mathbf{V}_I^\top$.

$\mathbf{Q}_1, \mathbf{R}_1 \leftarrow \text{QR}\left(\begin{bmatrix}\mathbf{U}_1\Sigma_1 & \mathbf{U}_2\Sigma_2\end{bmatrix}\right)$;
$\mathbf{Q}_2, \mathbf{R}_2 \leftarrow \text{QR}\left(\begin{bmatrix}\mathbf{V}_1 & 0 \\ 0 & \mathbf{V}_2^\top\end{bmatrix}\right)$;
$\mathbf{U}_R, \Sigma_R, \mathbf{V}_R^\top \leftarrow \text{SVD}(\mathbf{R}_1\mathbf{R}_2^\top)$;
$\mathbf{U}_I, \Sigma_I, \mathbf{V}_I^\top \leftarrow \mathbf{Q}_1\mathbf{U}_R, \Sigma_R, \mathbf{V}_R^\top\mathbf{Q}_2^\top$;
return $\mathbf{U}_I, \Sigma_I, \mathbf{V}_I^\top$;

	u_1	u_2
v_2	$\ln(t_1 + t_2)$	$\ln t_4$
v_3	$\ln t_2$	$\ln t_3$
v_5	0.0	$\ln t_4$

(a) Origin Temporal Matrix

	u_1	u_2	u_3
v_2	$\ln(t_1 + t_2)$	$\ln t_4$	0.0
v_3	$\ln t_2$	$\ln t_3$	$\ln t_5$
v_5	0.0	$\ln t_4$	$\ln(t_5 + t_6)$
v_6	0.0	0.0	$\ln t_6$

(b) Incremental Temporal Matrix

Figure 3: A toy example of incremental temporal matrix update (the left is the original temporal matrix of node v_2, v_3, v_5 in Figure 2, and the right is the new temporal matrix after adding edge (v_2, v_5) at time t_5 and (v_5, v_6) at time t_6.

Nevertheless, existing incremental SVD methods either need high computation cost [65] or have unique striction on matrix [26] to get an exact result. In this section, we propose an incremental SVD method with both high efficiency and theoretical bound to compute its truncated result. Specifically, for an original temporal matrix $\mathbf{W}$ and a new temporal matrix $\mathbf{X} = [\mathbf{W}, \mathbf{W}_i]$, where $\mathbf{W}_i \in \mathbb{R}^{n \times \beta}$ is the newly added data and we already have the svd $\mathbf{U}\Sigma\mathbf{V}^\top$ of $\mathbf{W}$, written as $\mathbf{W_d}$. Let the svd of $\mathbf{W}_i = \mathbf{U}_2\Sigma_2\mathbf{V}_2^\top$. First, a factorization of $\mathbf{X}$ could be written as:

$$\mathbf{X} = \begin{bmatrix}\mathbf{U}\Sigma & \mathbf{U}_2\Sigma_2\end{bmatrix}\begin{bmatrix}\mathbf{V}^\top & 0 \\ 0 & \mathbf{V}_2^\top\end{bmatrix}.$$

Next, we conduct QR factorization on each part of the above factorization to get the orthogonal submatrix with an upper triangular matrix in center. Finally, factorize the upper triangular matrix by SVD, then the approximate incremental SVD can be obtained and the new embeddings are the product of the left part and the singular value matrix in the center. The detail is shown in Algorithm 2.

The following lemma shows that the results returned from Algorithm 2 are equal to the SVD of new temporal matrix $\mathbf{X}$.

Lemma 5.1. *While* $\mathbf{W}$ *is the original temporal matrix, if* $\mathbf{U}_1\Sigma_1 = \mathbf{U}\Sigma$ *and* $\mathbf{V}_1 = \mathbf{V}$, *the* $\mathbf{U}_I, \Sigma_I, \mathbf{V}_I^\top$ *returned from Algorithm 2 is equal to the SVD of new temporal matrix* $[\mathbf{W}, \mathbf{W_i}]$.

Proof. Based on Algorithm 2, we can get

$$[\mathbf{W}, \mathbf{W}_i] = \mathbf{Q}_1\mathbf{U}_R\Sigma_R\mathbf{V}_R^\top\mathbf{Q}_2^\top.$$

Algorithm 3: LTGEInc

Input: Newly added edge set $\mathcal{E}'$, original embeddings $\mathbf{Z}$, original truncated right singular matrix $\mathbf{V}_d$, original number of snapshot k, parameter ρ, original time span T.

Output: New embeddings $\mathbf{Z}_i$, new right singular matrix $\mathbf{V}_i$.

$t_{\text{label}} \leftarrow 0$, snapshot $\leftarrow 0$, size $\leftarrow \frac{T}{k}$;
$k' \leftarrow |\mathcal{E}'|/\lfloor \frac{T}{k} \rfloor$;
Initialize a null matrix $\mathbf{W}_i \in \mathbb{R}^{n \times k'}$;
for $(v_i, v_j) \in \mathcal{E}'$ **do**
  if $t_{v_i,v_j} - t_{\text{label}} >$ size **then**
    $t_{\text{label}} \leftarrow t_{v_i,v_j}$;
    snapshot $\leftarrow$ snapshot + 1;
  $\mathbf{W}_i[v_i, \text{snapshot}] \leftarrow \mathbf{W}_i[v_i, \text{snapshot}] + t_{v_i,v_j}$;
  $\mathbf{W}_i[v_j, \text{snapshot}] \leftarrow \mathbf{W}_i[v_j, \text{snapshot}] + t_{v_i,v_j}$;
$\mathbf{W}_i \leftarrow \sqrt{\rho} \ln \mathbf{W}_i$ for all nonzero elements in $\mathbf{W}_i$;
$\mathbf{U}_2, \Sigma_2, \mathbf{V}_2^\top \leftarrow \text{TruncatedSVD}(\mathbf{W}_i, d)$;
$\mathbf{U}_i, \Sigma_i, \mathbf{V}_i^\top \leftarrow \text{IncrementalSVD}(\mathbf{Z}, \mathbf{V}_d^\top, \mathbf{U}_2\Sigma_2, \mathbf{V}_2^\top)$;
$\mathbf{Z}_i \leftarrow \mathbf{U}_i\Sigma_i$;
return $\mathbf{Z}_i, \mathbf{V}_i^\top$;

Since $\mathbf{Q}_1^\top\mathbf{Q}_1 = \mathbf{I}$ and $\mathbf{U}_R^\top\mathbf{U}_R = \mathbf{I}$, we have:

$$(\mathbf{Q}_1\mathbf{U}_R)^\top(\mathbf{Q}_1\mathbf{U}_R) = \mathbf{I}$$

$$[\mathbf{W}, \mathbf{W}_i]^\top[\mathbf{W}, \mathbf{W}_i] = (\mathbf{Q}_2\mathbf{V}_R)\Sigma_R^2(\mathbf{Q}_2\mathbf{V}_R)^\top.$$

Similarly, $(\mathbf{Q}_2\mathbf{V}_R)^\top(\mathbf{Q}_2\mathbf{V}_R) = \mathbf{I}$, so that:

$$[\mathbf{W}, \mathbf{W}_i]^\top[\mathbf{W}, \mathbf{W}_i](\mathbf{Q}_2\mathbf{V}_R) = (\mathbf{Q}_2\mathbf{V}_R)\Sigma_R^2.$$

Therefore, the eigenvalue of $[\mathbf{W}, \mathbf{W}_i]^\top[\mathbf{W}, \mathbf{W}_i]$ is Σ_R^2, which means Σ_R is also the singular value matrix of $[\mathbf{W}, \mathbf{W}_i]$ and $\mathbf{U}', \Sigma', \mathbf{V}'^\top$ returned from Algorithm 2 is the truncated SVD of $[\mathbf{W}, \mathbf{W}_i]$. Thus we finish the proof. □

Based on Lemma 5.1, we could set the truncated SVD of origin as input of incremental SVD to get the increment of $\mathbf{X}' = [(\mathbf{W})_d, (\mathbf{W}_i)_d]$, while the embeddings generated from LTGE satisfies that it is the left part of $(\mathbf{W})_d$'s SVD. LTGEInc takes the embeddings with truncated right singular matrix as input of Algorithm 2 and gets the final incremental embeddings. Thus it reduces the computational dimension of the incremental SVD. Algorithm 3 shows the whole process of LTGEInc.

Next, we will give the bound of approximate incremental SVD. For an original temporal matrix $\mathbf{W}$ and a new temporal matrix $\mathbf{X} = [\mathbf{W}, \mathbf{W}_i]$ that needs to be updated, the error between approximate incremental matrix $\widetilde{\mathbf{X}'}$ and original new temporal matrix $\mathbf{X}$ can be bounded by the following lemma:

Lemma 5.2. *Let $\widetilde{\mathbf{X}} = (\mathbf{X})_\mathbf{d}$, $\mathbf{X}' = [(\mathbf{W})_d, (\mathbf{W}_i)_d]$ and $\widetilde{\mathbf{X}'} = (\mathbf{X}')_\mathbf{d}$, where we have $\mathbf{W} \in \mathbb{R}^{n\times k}$, $\mathbf{W}_i \in \mathbb{R}^{n\times\beta}$ and $\mathbf{X}, \widetilde{\mathbf{X}}, \mathbf{X}', \widetilde{\mathbf{X}'} \in \mathbb{R}^{n\times(k+\beta)}$. The function $(\cdot)_d$ means approximating the matrix by preserving the first d singular values.*

The approximate incremental matrix $\widetilde{\mathbf{X}'}$ returned by Algorithm 3 have the following guarantee:

$$\|\widetilde{\mathbf{X}'} - \mathbf{X}\|_F \le 3\|\widetilde{\mathbf{X}} - \mathbf{X}\|_F.$$

Proof. Based on the triangle inequality of matrix norm (i.e., $\|\mathbf{A}+\mathbf{B}\| \le \|\mathbf{A}\| + \|\mathbf{B}\|$), we have

$$\begin{aligned}\|\widetilde{\mathbf{X}'} - \mathbf{X}\|_F &= \|\widetilde{\mathbf{X}'} - \mathbf{X}' + \mathbf{X}' - \mathbf{X}\|_F \\ &\le \|\widetilde{\mathbf{X}'} - \mathbf{X}'\|_F + \|\mathbf{X}' - \mathbf{X}\|_F.\end{aligned} \tag{5}$$

Meanwhile, since $\widetilde{\mathbf{X}'}$ is the d-rank approximation of $\mathbf{X}'$, we can get that

$$\|\widetilde{\mathbf{X}'} - \mathbf{X}'\|_F = \sqrt{\sum_{i=d+1}^{d+\beta}\sigma_i^2(\mathbf{X}')},$$

and based on the definition of Frobenius matrix norm (i.e., $\|\mathbf{A}\|_F = \sqrt{\sum_{i,j}\mathbf{A}[i,j]^2}$), we have

$$\begin{aligned}&\|\mathbf{X}' - \mathbf{X}\|_F = \\ &\sqrt{\sum_{i,j}(\mathbf{W}[i,j] - (\mathbf{W})_\mathbf{d}[i,j])^2 + \sum_{i,j}(\mathbf{W}_i[i,j] - (\mathbf{W}_i)_d[i,j])^2} \\ &\le \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{k}(\mathbf{W}[i,j] - (\mathbf{W})_d[i,j])^2} \\ &+ \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{\beta}(\mathbf{W}_i[i,j] - (\mathbf{W}_i)_d[i,j])^2} \\ &= \sqrt{\sum_{i=d+1}^{k}\sigma_i^2(\mathbf{W})} + \sqrt{\sum_{i=d+1}^{\beta}\sigma_i^2(\mathbf{W}_i)}.\end{aligned}$$

According to [65], for $\forall\mathbf{A}_1 \in \mathbb{R}^{n\times k}$ and $\forall\mathbf{A}_2 \in \mathbb{R}^{n\times\beta}$, it has:

$$\begin{aligned}\sigma_i(\mathbf{A}_1) &\le \sigma_i([\mathbf{A}_1, \mathbf{A}_2]), i = 1, 2, \ldots, k. \\ \sigma_i(\mathbf{A}_2) &\le \sigma_i([\mathbf{A}_1, \mathbf{A}_2]), i = 1, 2, \ldots, \beta. \\ \sigma_i([(\mathbf{A}_1)_d, (\mathbf{A}_2)_d]) &\le \sigma_i([\mathbf{A}_1, \mathbf{A}_2]), i = 1, 2, \ldots, k+\beta.\end{aligned}$$

So we have:

$$\begin{aligned}&\sigma_i(\mathbf{X}') \le \sigma_i([(\mathbf{W})_d, (\mathbf{W}_i)_d]) \le \sigma_i([\mathbf{W}, \mathbf{W}_i]) \le \sigma_i(\mathbf{X}), \\ &\sigma_i(\mathbf{W}) \le \sigma_i([\mathbf{W}, \mathbf{W}_i]) \le \sigma_i(\mathbf{X}), \\ &\sigma_i(\mathbf{W}_i) \le \sigma_i([\mathbf{W}, \mathbf{W}_i]) \le \sigma_i(\mathbf{X}).\end{aligned}$$

Then we can get:

$$\begin{aligned}\|\widetilde{\mathbf{X}'} - \mathbf{X}'\|_F &= \sqrt{\sum_{i=d+1}^{d+\beta}\sigma_i^2(\mathbf{X}')} \le \sqrt{\sum_{i=d+1}^{d+\beta}\sigma_i^2(\mathbf{X})} \\ &\le \sqrt{\sum_{i=d+1}^{k+\beta}\sigma_i^2(\mathbf{X})} = \|\widetilde{\mathbf{X}} - \mathbf{X}\|_F.\end{aligned} \tag{6}$$

Table 2: Time Complexity and Space Complexity.

Method	Time Complexity	Space
LTGE	$O(md\log(k) + nd^2\log(k))$	$O(m+nd)$
LTGEInc	$O(nd\beta + nd^2 + \Delta m)$	$O(nd + \Delta m)$

and

$$\begin{aligned}\|\mathbf{X}' - \widetilde{\mathbf{X}}\|_F &\le \sqrt{\sum_{i=d+1}^{k}\sigma_i^2(\mathbf{W})} + \sqrt{\sum_{i=d+1}^{\beta}\sigma_i^2(\mathbf{W_i})} \\ &\le \sqrt{\sum_{i=d+1}^{k}\sigma_i^2(\mathbf{X})} + \sqrt{\sum_{i=d+1}^{\beta}\sigma_i^2(\mathbf{X})} \\ &\le 2\sqrt{\sum_{i=d+1}^{k+\beta}\sigma_i^2(\mathbf{X})} = 2\|\widetilde{\mathbf{X}} - \mathbf{X}\|_F.\end{aligned} \tag{7}$$

Combining (6)–(7) completes the proof. □

Lemma 5.2 provides a theoretical guarantee for approximate incremental updating. The experiments on incremental future link prediction detailed in Section 7.5 corroborates that our method can maintain relatively stable performance while undergoing numerous updates, thus obviating the need for frequent recomputing to guarantee the quality of the embeddings. Concurrently, we analyze the complexity of the incremental algorithm in Section 6, showing that it is more efficient than recomputing.

6 COMPLEXITY ANALYSIS

In this section, we will analyze the complexity of LTGE and the incremental algorithm LTGEInc to explain their theoretical scalability. Table 2 shows the result of the time complexity of each method.

First, we give the analysis of LTGE (Algorithm 1). Constructing the temporal matrix $\mathbf{W}$ needs to operate on every edge in the original temporal graph G (lines 1–8) and perform log operation on every non-zero element (line 9). Since there are m edges in G and at most $2m$ non-zero elements in $\mathbf{W}$ as per our analysis in Section 4.2, the construction of $\mathbf{W}$ requires $O(m)$ time. Moreover, by Randomized SVD, factorizing $\mathbf{W} \in \mathbb{R}^{n\times k}$ with $O(m)$ non-zero elements to $\mathbf{U}_d \in \mathbb{R}^{n\times d}$, $\Sigma_d \in \mathbb{R}^{d\times d}$ and $\mathbf{V}_d \in \mathbb{R}^{n\times d}$ takes $O(md\log(k) + nd^2\log(k))$ [42] (lines 10). Finally, calculating $\mathbf{Z} = \mathbf{U}_d\Sigma_d \in \mathbb{R}^{n\times d}$ takes $O(nd^2)$ time (lines 11). Putting it together, the total time complexity of LTGE is $O(md\log(k) + nd^2\log(k))$. Meanwhile, it takes $O(nd)$ to store $\mathbf{Z}$, $\mathbf{U}_d$, Σ_d and $\mathbf{V}_d$, while storing $\mathbf{W}$ with $O(m)$ non-zero elements takes $O(m)$ space via compressed sparse row [4]. Thus, the space complexity of LTGE is $O(m+nd)$.

Next, we analyze the IncrementalSVD algorithm (Algorithm 2). For $\mathbf{U}_1\Sigma_1, \mathbf{V}_1, \mathbf{U}_2\Sigma_2, \mathbf{V}_2 \in \mathbb{R}^{n\times d}$, performing QR factorization takes $O(nd^2)$ time [56] (lines 1–2). For $\mathbf{R}_1, \mathbf{R}_2 \in \mathbb{R}^{d\times d}$, performing SVD of $\mathbf{R}_1\mathbf{R}_2$ takes $O(d^3)$ time (lines 3). Finally, for $\mathbf{Q}_1, \mathbf{Q}_2 \in \mathbb{R}^{n\times d}$ and $\mathbf{U}_R, \mathbf{V}_R \in \mathbb{R}^{d\times d}$, calculating $\mathbf{U}_I = \mathbf{Q}_1\mathbf{U}_R$ and $\mathbf{V}_I^\top = \mathbf{V}_R^\top\mathbf{Q}_2^\top$ takes $O(nd^2)$ time (line 4). Therefore, the time complexity of Algorithm 2 is $O(nd^2)$. Meanwhile, its space complexity is $O(nd)$.

Now, we derive the complexity of LTGEInc (Algorithm 3) when the newly added data span β snapshots. Constructing $\mathbf{W}_i$ with at most Δm non-zero elements needs $O(\Delta m)$ time (lines 1–10). Performing truncated SVD on $\mathbf{W}_i \in \mathbb{R}^{n\times\beta}$ takes $O(nd\beta)$ time [53] (line 11). Invoking Algorithm 2 takes $O(nd^2)$ time (line 12). For $\mathbf{U}_i \in \mathbb{R}^{n\times d}$ and $\Sigma_i \in \mathbb{R}^{d\times d}$, calculating $\mathbf{Z}_i = \mathbf{U}_i\Sigma_i$ takes $O(nd^2)$ time (lines 13). Therefore, the total time complexity of LTGEInc is $O(nd\beta + nd^2 + \Delta m)$. Meanwhile, LTGEInc takes $O(nd)$ to store matrices except for $\mathbf{W}_i$ which requires $O(\Delta m)$ space. Therefore, the space complexity of LTGEInc is $O(nd + \Delta m)$.

7 EXPERIMENTS

We evaluate our proposed method LTGE and LTGEInc, by comparing them with 11 baselines across three tasks: future link prediction, future top-K recommendation and incremental future link prediction. Experiments are conducted on a single machine with Intel Xeon 8377C, 1T RAM. For baselines that need GPU, we use one Nvidia 3090.

7.1 Datasets

In this paper, ten public datasets are used to widely evaluate the performance of LTGE and LTGEInc. The datasets contain various kinds of temporal graphs, including social graphs, web graphs, rating graphs, shopping graphs, etc. The statistics of each dataset are listed in Table 3. CollegeMsg[1] [44] is a private message social graph and each edge represents a message sent between two users. Bitcoin[1] [32, 33] is a user-trust-user graph, where edges represent users' trust in blockchain transactions. Serendipity[2] [31] is a bipartite rating graph that collects the rating when user meets serendipitous recommendation. Math[1], AskUbuntu[1], Superuser[1] and Stack[1] are all web graphs, in which nodes represent users and edges means question and answer between users [45]. Movielen[2] [20] is a rating graph collected from the MovieLen web site. Wikitalk[1] [35, 45] is an editing graph representing Wikipedia users editing each other's Talk page. Taobao[3] [69, 71] is a shopping graph that we generate from the Tianchi, which contains over 17 million nodes and 1.3 billion edges. To our best knowledge, there is no existing work on temporal graph embedding can deal with datasets on this scale.

7.2 Baselines and Parameter Settings

We compare LTGE with 11 competitive methods, including (i) four temporal graph embedding methods: Zebra [37], APAN [59], TGN [51], and CTDNE [43], (ii) five simple graph embedding methods for large-scale graphs: Geep [60], HuGE [15], ProNE [67], LINE [55], node2vec [18], and (iii) three dynamic graph embedding methods: DAMF[13], GloDyNE [22], LocalAction [38].

For fair comparison, we set the embedding dimension d as 32 for all methods and implement them in Python. The iteration number of randomized svd is fixed to 6 to get best performance for every dataset. We select the parameter k from $\{100, 200, 400, 800, 1600\}$ and select ρ from $\{0.001, 0.01, 0.1, 1, 10\}$ in LTGE.

The results of future link prediction, incremental link prediction and future top-K recommendation are reported in Sections 7.3, 7.5 and 7.6, respectively. We eliminate methods that cannot finish in

[1] https://snap.stanford.edu/data/
[2] https://grouplens.org/datasets/
[3] https://tianchi.aliyun.com/dataset/140281/

Table 3: Statistics of Datasets. Task 1: Link Prediction, Task 2: Recommendation, Task 3: Incremental Link Prediction. (K = 10^3, M = 10^6, B = 10^9)

Name	$\|V\|$	$\|E\|$	$\|T\|$	Task
CollegeMsg	1.9K	59.8K	193 days	1
Bitcoin	5.8K	33.5K	1903 days	1 & 2
Math	24.8K	506.5K	2350 days	1
Serendipity	153.6K	9.9M	2978 days	2
AskUbuntu	159.3K	964.4K	2613 days	1
Superuser	194.0K	1.4M	2773 days	1
Movielen	221.6K	25M	9082 days	2
Wikitalk	1.1M	7.8M	2320 days	1 & 3
Stack	2.6M	63.4M	2774 days	1 & 3
Taobao	17.9M	1.3B	184 days	1 & 2 & 3

three days or run out of memory. In Section 7.4, we evaluate the efficiency of all methods and the scalability of LTGE. Finally, the evaluation results of LTGE with different parameters are reported in Section 7.7.

7.3 Future Link Prediction

Future link prediction is a fundamental downstream task for temporal graph analysis. It aims to predict whether an edge is likely to form between nodes in the future. For all datasets, we first sort all edges by increasing order with their timestamp, split them into 70%-30% for obtaining the embeddings and testing. Following [70], since we do not use node features or edge features for all methods, we remove the test edges that contain unseen nodes in training set for correctness. Then we randomly sample the same number of negative edges for testing. To get the prediction score of edges, LTGE first obtains the embeddings using positive training edges. Then it mixes positive test edges and negative test edges to generate a whole test set. For each edge (v_i, v_j, t) in the test set, we use the sigmoid of dot product $z_{v_i} \cdot z_{v_j}$ as the prediction score.

Following prior work [29, 37], Area Under Curve (AUC) and Average Precision (AP) are adopted to evaluate future link prediction performance. Table 4 reports the result of LTGE and other competitors on each dataset. The highest score is highlighted with blue and the second best is single-underlined. It shows that LTGE outperforms all competitors over all datasets for all metrics. On AskUbuntu, LTGE achieves 91.0% AUC and 93.8% AP, outperforming the best-performing competitor Zebra by a margin of up to 1.4% for AUC score and 2.5% for AP score. On the largest dataset Taobao, only three simple graph embedding method and LTGE can successfully generate the graph embeddings, while LTGE achieve 77.3% AUC and 79.1% AP, significantly superior than the strongest competitor DAMF with 10.3% improvement in AUC and 9.7% improvement in AP. These findings underscore LTGE's superior performance and efficiency, which is due to the fact that LTGE properly defines a proper temporal similarity TPASim and generates the embedding that is equivalent with factoring this similarity matrix. TPASim takes into account the association between the nodes and different snapshots, thus being able to preserve both temporal information and structural information. To sum up, LTGE demonstrates an average improvement of 3.82% in the AUC score and 4.01% in the AP score compared to the second best baseline while it achieves two orders of magnitude faster compared to the strongest temporal graph embedding method Zebra.

7.4 Efficiency and Scalability

In terms of efficiency, we conduct a comprehensive analysis to evaluate the performance of various methods used for generating embeddings in future link prediction. Figure 4 illustrates the running time of each method, excluding data preprocessing, loading edges, and outputting embeddings. Notably, for HuGE, the computation of common neighbors is considered part of the algorithm rather than preprocessing, so it is included in its overall running time.

As depicted in Figure 4, LTGE surpasses all competitors in terms of efficiency across all datasets. Temporal graph neural network based methods generally outperform simple graph methods in terms of effectiveness, but their efficiency lags behind. LTGE is more efficient than existing temporal graph embedding methods. For example, it takes only 9.18 seconds for LTGE to generate the embeddings for AskUbuntu, 116× faster than Zebra that needs 1071.70 seconds to train the T-GNN model, while LTGE has the higher AUC and AP score. For massive temporal graph Stack and Taobao, it is difficult for most existing temporal graph embedding methods to finish in three days without running out of memory. This situation reflects that LTGE fills the gap by providing an efficient method for large temporal graph embedding. In particular, on the largest dataset Taobao, only LTGE, ProNE, DAMF and LocalAction could generate embeddings in three days, while only LTGE specially designs for temporal graphs and considers temporal information. Compared with the strongest temporal graph embedding method Zebra, LTGE is two orders of magnitude faster, demonstrating its superior efficiency and capability to handle massive temporal graphs.

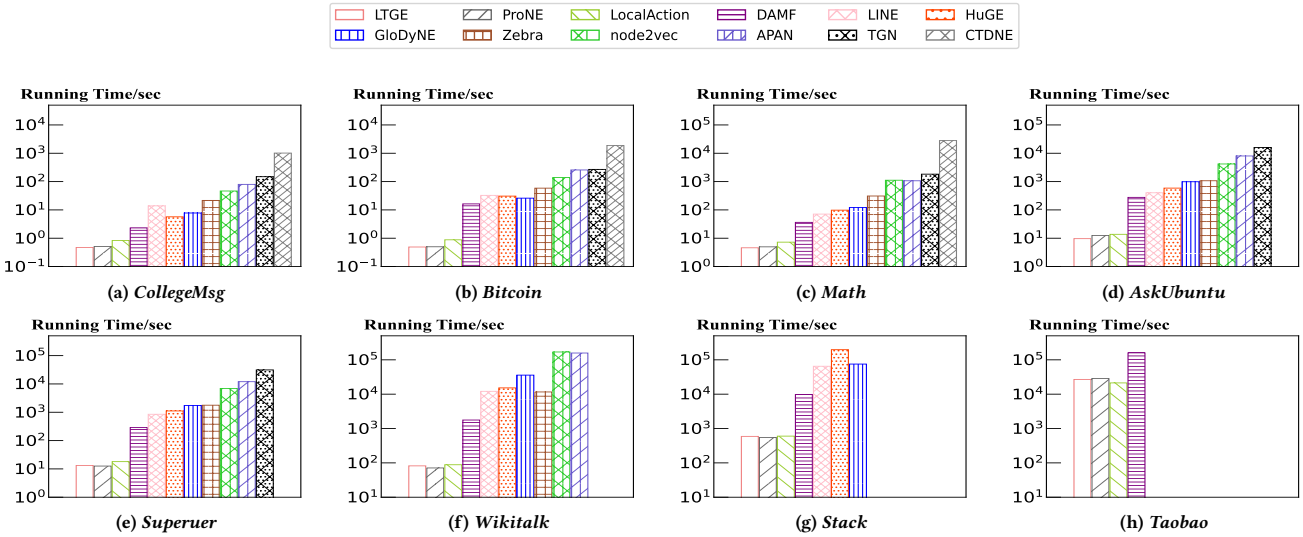
To further evaluate the scalability of LTGE, we conduct tests on the graphs with varying scales. Following previous work [60], we utilize the random graph model from [14] to generate random graphs of different sizes and record the running times of LTGE. For comparison purposes, we also include the running time of the most efficient T-GNN model Zebra.

In Figure 5a, we keep the number of edges at 1×10^7 and vary the number of nodes in $\{2 \times 10^5, 4 \times 10^5, 6 \times 10^5, 8 \times 10^5, 1 \times 10^6\}$. The plots show that the running time of LTGE and Zebra increases below linear growth with the number of nodes, with LTGE exhibiting a slow growth. Similarly, in Figure 5b, we keep the number of nodes at 1×10^6 and varied the number of edges in $\{1 \times 10^7, 2 \times 10^7, 3 \times 10^7, 4 \times 10^7, 5 \times 10^7\}$. LTGE and Zebra display near-linear growth in running time, and LTGE also exhibits a slower growth showing that the constant factor is far more smaller than Zebra. This observation suggests that the running time of LTGE is more sensitive to the increase in the number of edges. Overall, the scalability tests demonstrate that our proposed method LTGE is scalable and consistent with the presented time complexity analysis.

It is worth mentioning that due to the inherent trade-off between efficiency and effectiveness in incremental updating tasks, we will

Table 4: Future link prediction performance.

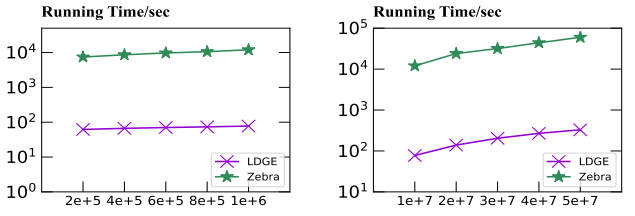
Method	*CollegeMsg*		*Bitcoin*		*Math*		*AskUbuntu*		*Superuser*		*Wikitalk*		*Stack*		*Taobao*	
	AUC	AP	AUC	AP	AUC	AP	AUC	AP	AUC	AP	AUC	AP	AUC	AP	AUC	AP
node2vec	0.642	0.646	0.624	0.612	0.718	0.717	0.665	0.696	0.651	0.701	0.611	0.580	-	-	-	-
LINE	0.601	0.640	0.511	0.582	0.830	0.880	0.737	0.811	0.720	0.801	0.756	0.801	0.601	0.614	-	-
ProNE	0.764	0.772	0.622	0.606	0.893	0.902	0.814	0.858	0.792	0.831	0.896	0.903	0.858	0.856	0.613	0.632
HuGE	0.801	0.810	0.769	0.776	0.920	0.929	0.868	0.895	0.852	0.871	0.923	0.931	0.879	0.887	-	-
CTDNE	0.502	0.533	0.560	0.648	-	-	-	-	-	-	-	-	-	-	-	-
TGN	0.830	0.834	0.858	0.878	0.890	0.907	0.867	0.892	0.828	0.846	-	-	-	-	-	-
APAN	0.800	0.805	0.832	0.849	0.905	0.914	0.860	0.883	0.819	0.838	-	-	-	-	-	-
Zebra	0.827	0.830	0.864	0.883	0.941	0.950	0.896	0.913	0.888	0.907	0.963	0.972	-	-	-	-
LocalAction	0.578	0.588	0.536	0.559	0.595	0.607	0.544	0.552	0.431	0.460	0.613	0.672	0.608	0.649	0.501	0.546
GloDyNE	0.642	0.651	0.646	0.641	0.720	0.721	0.632	0.640	0.640	0.658	0.580	0.556	0.759	0.762	-	-
DAMF	0.801	0.809	0.853	0.879	0.922	0.933	0.835	0.871	0.845	0.876	0.945	0.954	0.856	0.876	0.670	0.694
LTGE	0.852	0.853	0.878	0.905	0.955	0.965	0.910	0.938	0.912	0.931	0.966	0.977	0.919	0.939	0.773	0.791



(a) *CollegeMsg* (b) *Bitcoin* (c) *Math* (d) *AskUbuntu*

(e) *Superuer* (f) *Wikitalk* (g) *Stack* (h) *Taobao*

Figure 4: Running time (best viewed in color).



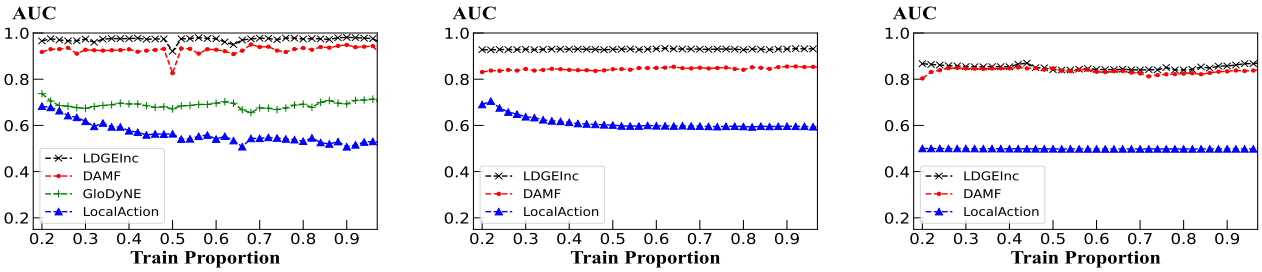
(a) Running time varying #nodes. (b) Running time varying #edges.

Figure 5: Running time with different scale.

discuss the efficiency analysis of LTGEInc alongside its effects in Section 7.5.

7.5 Incremental Link Prediction

As mentioned in Section 5, incremental learning is an important ability that is always ignored by existing temporal graph embedding works. It aims at learning from newly added edges and nodes continuously and then updating embeddings (or updating GNN model). It is meaningless to conduct the incremental experiments on small datasets since we can get accurate result by recomputing the embeddings with little cost. For large datasets including Wikitalk, Stack and Taobao, we first use only 20% percent of edges to obtain the embeddings and test future link prediction on the following 2% percent of edges. Then the positive edges in the test set will be added into training set, and the test set will change to the next 2% future edges. The process will be repeated until the end of whole edge set to simulate data growth that appears in real-world scenarios.



(a) **Wikitalk** (b) **Stack** (c) **Taobao**

Figure 6: Incremental link prediction results (best viewed in color).

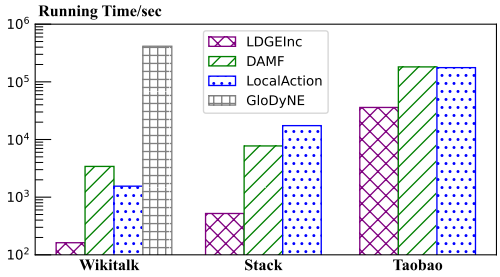


Figure 7: Running time of incremental link prediction

Table 5: Running time of recomputation vs. LTGEInc.

Datasets	Recomputation	LTGEInc
Wikitalk	90.2s	4.4s
Stack	672.7s	31.8s
Taobao	29128.6s	1163.2s

We choose DAMF, LocalAction and GloDyNE as baselines in this task. To measure the performance of incremental link prediction task, AUC score is used for every set of tests during the evaluation. We eliminate methods that cannot finish in five days or run out of memory.

Figure 6 shows the results of LTGEInc and competitors. Particularly in the Wikitalk dataset, LTGEInc consistently delivers the highest AUC performance in every test. In contrast, LocalAction and GloDyNE exhibit a varying degree of performance deterioration with new graph data. DAMF exhibits less susceptibility to degradation but it always performs worse than LTGEInc.

Table 5 gives the running time of recomputing and LTGEInc on each dataset when facing the last 2% test edges. It is obvious that LTGEInc can speed up more than an order of magnitude compared with recomputing the whole embeddings. Also, Figure 7 gives the whole running time during the incremental link prediction task. LTGEInc achieves the shortest running time for each dataset, which is based on the rapid incremental updating algorithm. At the same time, it also shows that LTGEInc demonstrates remarkable stability across all three test datasets, outperforming all baselines under nearly all scenes. This performance underlines the effectiveness of our uniquely designed incremental embedding update method in handling new data processing challenges.

7.6 Future Top-K Recommendation

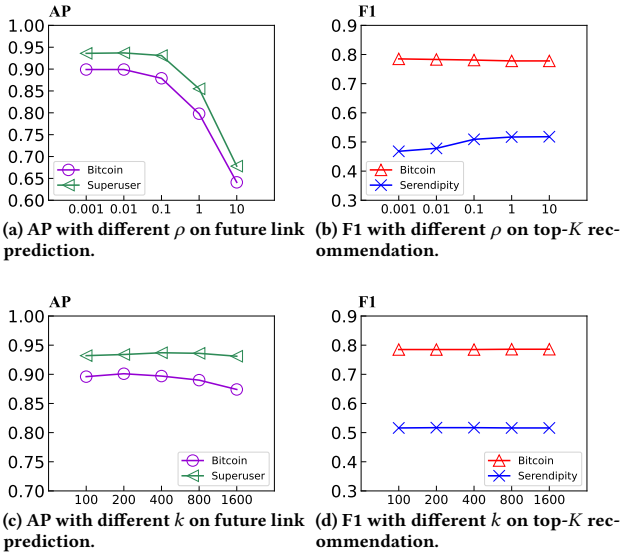
Top-K recommendation is another usual downstream task, where the objective is to recommend K nodes to each node$u$. For instance, given a user-good graph G, the target is to recommend K items potentially of interest to each user u. In a temporal graph context, each edge appears as serialized data, making it ill-suited to random test set selection based on simple graph testing methods. Hence, we examine the effect of the future top-K recommendation task. Specifically, edges are sorted in ascending order by their generation time. The first 70% of these time-ordered edges are used as the training set for embeddings generation, while the remaining edges are served as ground-truth for testing. For each node v_i appearing in the test set, we first generate all nodes connected to node v_i in the test set, then use the dot product $z_{v_i} \cdot z_{v_j}$ to calculate the similarity between each node v_i and node v_j, identifying the K nodes with the greatest bias to node v_i as a recommendation list.

Next, we generate the actual top-K list of node v_i based on the ground-truth list in the test set. Comparing the ground-truth list and recommendation list for each node, we compute four standard metrics: **F1**, Mean Average Precision (**MAP**), Normalized Discounted Cumulative Gain (**NDCG**), and Mean Reciprocal Rank (**MRR**). All of the four metrics are the higher the better and we report the average scores across all test nodes. We'll exclude the edges in the test set containing unseen nodes or missing ground truth to ensure the correctness. Because T-GNNs are end-to-end and principally designed for future link prediction or future edge classification, they are not well suited to future top-K recommendation task. Thus, we will not compare with T-GNN related methods in this task. Since Bitcoin is not a bipartite graph, we do not apply Geep to this dataset. Meanwhile, any method that runs for more than three days or out of memory will be terminated.

Table 6 shows the result of future top-K (K=10) recommendation. LTGE also outperforms all competitors on all the datasets in this task. For instance, LTGE achieves 41.4% F1, 31.6% MAP, 62.3% MRR and 42.6% NDCG on Movielen, where all metrics are higher than the strongest competitor LocalAction with 7.0% up to F1, 5.0% up to MAP, 10.1% up to MRR and 5.8% up to NDCG. On the non-bipartite graph Bitcoin, LTGE still performs well on this type of graph and beats all baselines for all metrics. In summary, results show that LTGE also performs well on future top-K recommendation tasks. The excellent performance in future link prediction and top-K recommendation tasks simultaneously reflects the universality of LTGE.

Table 6: Future top-K ($K = 10$) recommendation performance.

Method	*Bitcoin*				*Serendipity*				*Movielen*				*Taobao*			
	F1	MAP	MRR	NDCG	F1	MAP	MRR	NDCG	F1	MAP	MRR	NDCG	F1	MAP	MRR	NDCG
node2vec	0.773	0.665	0.916	0.742	0.440	0.353	0.573	0.437	-	-	-	-	-	-	-	-
LINE	0.779	0.670	0.921	0.748	0.440	0.352	0.567	0.436	0.319	0.238	0.442	0.315	-	-	-	-
ProNE	0.776	0.666	0.915	0.744	0.443	0.356	0.581	0.441	0.324	0.243	0.458	0.322	0.731	0.675	0.808	0.732
HuGE	0.781	0.673	0.925	0.751	0.444	0.356	0.584	0.442	0.316	0.237	0.442	0.315	-	-	-	-
Geep	-	-	-	-	0.459	0.367	0.622	0.461	0.333	0.250	0.492	0.335	0.730	0.674	0.804	0.731
CTDNE	0.777	0.670	0.915	0.746	-	-	-	-	-	-	-	-	-	-	-	-
GloDyNE	0.772	0.663	0.913	0.740	0.441	0.353	0.566	0.437	0.322	0.241	0.448	0.319	-	-	-	-
LocalAction	0.776	0.673	0.918	0.747	0.459	0.367	0.605	0.458	0.364	0.276	0.522	0.368	0.751	0.702	0.820	0.753
DAMF	0.782	0.674	0.922	0.752	0.464	0.371	0.627	0.466	0.337	0.254	0.502	0.340	0.786	0.751	0.850	0.791
LTGE	0.785	0.679	0.941	0.758	0.518	0.422	0.713	0.528	0.414	0.316	0.623	0.426	0.887	0.882	0.975	0.911



(a) AP with different ρ on future link prediction.

(b) F1 with different ρ on top-K recommendation.

(c) AP with different k on future link prediction.

(d) F1 with different k on top-K recommendation.

Figure 8: Results with different parameters.

7.7 Parameter Analysis

In this section, we conduct experiments on Bitcoin, Serendipity and Superuser to study the impact of different parameters used in LTGE and give advice for choosing the applicable parameters. In short, the setting of parameter ρ depends on different downstream tasks and types of graphs. For future link prediction, the guiding setting is $\rho = 0.001$. For top-K recommendation, $\rho = 0.001$ is also suitable for non-bipartite graph and $\rho = 1$ is a good choice for bipartite graph. For the number of snapshots k, the choice of it should depend on the characteristics of different temporal graphs, but it will not significantly affect the performance within the reasonable range from 100 to 1600 as we suggest.

Impact of parameter ρ. To study the impact of ρ on LTGE, we fix the parameter k to 400 and report the performance with varying ρ in $\{0.001, 0.01, 0.1, 1, 10\}$ on future link prediction and top-K recommendation task. As shown in Figure 8a, AP on each dataset has increasing with smaller ρ until $\rho = 10^{-3}$ in future link prediction task. On top-K recommendation task, the F1 score increases when ρ become larger and is basically stable for $\rho \geq 1$ for bipartite graph Serendipity. For non-bipartite graph Bitcoin, $\rho = 10^{-3}$ is the best choice. Through the above results, we suggest choosing $\rho = 10^{-3}$ for future link prediction and top-K recommendation on non-bipartite graphs. For bipartite graphs, $\rho = 10^{-3}$ is also suitable for future link prediction but $\rho = 1$ should be used on top-K recommendation.

Impact of the number of snapshots k. In Lemma 4.2, we prove that TPA will degrade into a static similarity when $k = 1$, which indicates that choosing too small k may make the embeddings miss temporal information. On the other hand, too large k will split a large amount of snapshots, while each snapshot contains only a few edges. It will not only reduce the efficiency of LTGE but damage the quality of embeddings. To investigate the appropriate choice of k, we fix ρ for different downstream tasks and datasets as we suggest above and select k from $\{100, 200, 400, 800, 1600\}$. Figure 8c and 8d report the results with different k. On future link prediction task, the performance of LTGE decreases slightly with the increasing of k and also decreases with k smaller than 200, which is consistent with our analysis. Also, the best performance shows on k smaller than 200 for small dataset Bitcoin and k larger than 400 for medium dataset Superuser, which indicates that large datasets need large k. On top-K recommendation task, the change of k in our given range will not significantly affect the F1 score. In general, it is reasonable to select k as 200 to small datasets and increase it for large datasets, which will not make appreciable impact for performance in this reasonable range.

To sum up, LTGE is easy to achieve good performance on various datasets with the parameters we suggest and it is not necessary to put a lot of effort to adjust parameters if anyone uses LTGE for different datasets. For LTGEInc, it has the same conclusion since the parameters of LTGEInc is consist with LTGE.

8 CONCLUSION AND DISCUSSION

In this paper, we introduce a large temporal graph embedding method LTGE that can efficiently encode temporal information in temporal graphs. Specifically, we propose a new data structure

temporal-based bipartite graph (TBG) to model connections between nodes and temporal information. Then we devise an innovative node similarity measure called temporal distribution similarity (TPASim) and efficiently generate the high-quality embeddings with the property of TPASim. In addition, an advanced incremental learning algorithm LTGEInc is designed, offering an effective solution with theoretical guarantee to the challenges of incremental embedding update in real-world temporal graphs. Extensive experiments demonstrate that our methods LTGE and LTGEInc surpass state-of-the-art solutions in both efficiency and effectiveness.

In current work, our method does not consider node features, which may further enhance the embedding quality as evidenced by previous T-GNNs like Zebra. For such attributed graphs, a naive solution is to fuse the feature information by concatenating the node features with the embedding generated by LTGE. In our future work, we aim to expand our methods to attributed graphs by incorporating the type of nodes and edges into the embeddings. In addition, when graphs evolves, LTGEInc focuses on the addition of edges and nodes, leaving room for handling the deletion of edges and nodes. Although some existing work [13, 72] on incremental SVD for matrix deletion can be used to solve this problem, it does not have a provable guarantee, making LTGEInc less effective in cases where nodes or edges need to be deleted frequently. In the future, we will focus on designing theoretically guaranteed algorithms for scenarios involving node and edge deletion. explore the potential use of LTGE in other areas such as graph anomaly detection and temporal graph generation.

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

[3] Mikhail Belkin and Partha Niyogi. 2001. Laplacian eigenmaps and spectral techniques for embedding and clustering. *Advances in neural information processing systems* 14 (2001).

[4] Aydin Buluç, Jeremy T Fineman, Matteo Frigo, John R Gilbert, and Charles E Leiserson. 2009. Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. In *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*. 233–244.

[5] Ines Chami, Adva Wolf, Da-Cheng Juan, Frederic Sala, Sujith Ravi, and Christopher Ré. 2020. Low-Dimensional Hyperbolic Knowledge Graph Embeddings. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 6901–6914.

[6] Xu Chen, Siheng Chen, Jiangchao Yao, Huangjie Zheng, Ya Zhang, and Ivor W Tsang. 2020. Learning on attribute-missing graphs. *IEEE transactions on pattern analysis and machine intelligence* 44, 2 (2020), 740–757.

[7] Xiaolong Chen, Yifan Song, and Jing Tang. 2024. Link Recommendation to Augment Influence Diffusion with Provable Guarantees. *arXiv preprint arXiv:2402.19189* (2024).

[8] Yuhan Chen, Yihong Luo, Jing Tang, Liang Yang, Siya Qiu, Chuan Wang, and Xiaochun Cao. 2023. LSGNN: towards general graph neural network in node classification by local similarity. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*. 3550–3558.

[9] da Xu, chuanwei ruan, evren korpeoglu, sushant kumar, and kannan achan. 2020. Inductive representation learning on temporal graphs. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rJeW1yHYwH

[10] Quanyu Dai, Xiao Shen, Liang Zhang, Qiang Li, and Dan Wang. 2019. Adversarial training methods for network embedding. In *The World Wide Web Conference*. 329–339.

[11] 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*. https://doi.org/10.1145/3580305.3599250

[12] 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 *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*. https://doi.org/10.24963/ijcai.2018/288

[13] Xinyu Du, Xingyi Zhang, Sibo Wang, and Zengfeng Huang. 2023. Efficient Tree-SVD for Subset Node Embedding over Large Dynamic Graphs. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–26.

[14] László Erdős, Antti Knowles, Horng-Tzer Yau, and Jun Yin. 2013. SPECTRAL STATISTICS OF ERDŐS–RÉNYI GRAPHS I: LOCAL SEMICIRCLE LAW. *The Annals of Probability* (2013), 2279–2375.

[15] Peng Fang, Fang Wang, Zhan Shi, Hong Jiang, Dan Feng, and Lei Yang. 2021. HuGE: An entropy-driven approach to efficient and scalable graph embeddings. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2045–2050.

[16] Gene H Golub and Charles F Van Loan. 2013. *Matrix computations*. JHU press.

[17] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. 2018. DynGEM: Deep Embedding Method for Dynamic Graphs. (May 2018).

[18] 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.

[19] 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.

[20] F Maxwell Harper and Joseph A Konstan. 2015. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)* 5, 4 (2015), 1–19.

[21] Petter Holme, Beom Jun Kim, Chang No Yoon, and Seung Kee Han. 2002. Attack vulnerability of complex networks. *Physical review E* 65, 5 (2002), 056109.

[22] Chengbin Hou, Han Zhang, Shan He, and Ke Tang. 2022. GloDyNE: Global Topology Preserving Dynamic Network Embedding. *IEEE Transactions on Knowledge and Data Engineering* (Oct 2022), 4826–4837. https://doi.org/10.1109/tkde.2020.3046511

[23] Jie Huang, Chuan Chen, Fanghua Ye, Weibo Hu, and Zibin Zheng. 2020. Nonuniform hyper-network embedding with dual mechanism. *ACM Transactions on Information Systems (TOIS)* 38, 3 (2020), 1–18.

[24] Junjie Huang, Huawei Shen, Liang Hou, and Xueqi Cheng. 2021. SDGNN: Learning node representation for signed directed networks. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 35. 196–203.

[25] Shixun Huang, Zhifeng Bao, Guoliang Li, Yanghao Zhou, and J Shane Culpepper. 2020. Temporal network representation learning via historical neighborhoods aggregation. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 1117–1128.

[26] Mark A Iwen and BW Ong. 2016. A distributed and incremental SVD algorithm for agglomerative data analysis on large networks. *SIAM J. Matrix Anal. Appl.* 37, 4 (2016), 1699–1718.

[27] Glen Jeh and Jennifer Widom. 2002. Simrank: a measure of structural-context similarity. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. 538–543.

[28] Glen Jeh and Jennifer Widom. 2003. Scaling personalized web search. In *Proceedings of the 12th international conference on World Wide Web*. 271–279.

[29] Ming Jin, Yuan-Fang Li, and Shirui Pan. 2022. Neural temporal walks: Motif-aware representation learning on continuous-time dynamic graphs. *Advances in Neural Information Processing Systems* 35 (2022), 19874–19886.

[30] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42, 8 (2009), 30–37. https://doi.org/10.1109/MC.2009.263

[31] Denis Kotkov, Joseph A Konstan, Qian Zhao, and Jari Veijalainen. 2018. Investigating serendipity in recommender systems based on real user feedback. In *Proceedings of the 33rd annual acm symposium on applied computing*. 1341–1350.

[32] Srijan Kumar, Bryan Hooi, Disha Makhija, Mohit Kumar, Christos Faloutsos, and VS Subrahmanian. 2018. Rev2: Fraudulent user prediction in rating platforms. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 333–341.

[33] Srijan Kumar, Francesca Spezzano, VS Subrahmanian, and Christos Faloutsos. 2016. Edge weight prediction in weighted signed networks. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*. IEEE, 221–230.

[34] Mengyuan Lee, Guanding Yu, and Geoffrey Ye Li. 2020. Graph embedding-based wireless link scheduling with few training samples. *IEEE Transactions on Wireless Communications* 20, 4 (2020), 2282–2294.

[35] Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. 2010. Governance in social media: A case study of the Wikipedia promotion process. In *Proceedings of the International AAAI Conference on Web and Social Media*, Vol. 4. 98–105.

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

[37] Yiming Li, Yanyan Shen, Lei Chen, and Mingxuan Yuan. 2023. Zebra: When Temporal Graph Neural Networks Meet Temporal Personalized PageRank. *Proceedings of the VLDB Endowment* 16, 6 (2023), 1332–1345.

[38] 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*. https://doi.org/10.1145/3308560.3316585

[39] 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*. https://doi.org/10.1145/3308560.3316585

[40] 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.

[41] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[42] Cameron Musco and Christopher Musco. 2015. Randomized block krylov methods for stronger and faster approximate singular value decomposition. *Advances in neural information processing systems* 28 (2015).

[43] 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.

[44] Pietro Panzarasa, Tore Opsahl, and Kathleen M Carley. 2009. Patterns and dynamics of users' behavior and interaction: Network analysis of an online community. *Journal of the American Society for Information Science and Technology* 60, 5 (2009), 911–932.

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] Andrea Rossi, Donatella Firmani, Paolo Merialdo, and Tommaso Teofili. 2022. Explaining link prediction systems based on knowledge graph embeddings. In *Proceedings of the 2022 international conference on management of data*. 2062–2075.

[51] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. 2020. Temporal Graph Networks for Deep Learning on Dynamic Graphs. arXiv:2006.10637 [cs.LG]

[52] Sam T Roweis and Lawrence K Saul. 2000. Nonlinear dimensionality reduction by locally linear embedding. *science* 290, 5500 (2000), 2323–2326.

[53] Serge L Shishkin, Arkadi Shalaginov, and Shaunak D Bopardikar. 2019. Fast approximate truncated SVD. *Numerical Linear Algebra with Applications* 26, 4 (2019), e2246.

[54] Junwei Su, Difan Zou, Zijun Zhang, and Chuan Wu. 2023. Towards robust graph incremental learning on evolving graphs. In *International Conference on Machine Learning*. PMLR, 32728–32748.

[55] 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.

[56] Lloyd N Trefethen and David Bau. 2022. *Numerical linear algebra*. SIAM.

[57] 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.

[58] Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. 2018. Billion-scale commodity embedding for e-commerce recommendation in alibaba. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 839–848.

[59] Xuhong Wang, Ding Lyu, Mengjian Li, Yang Xia, Qi Yang, Xinwen Wang, Xinguang Wang, Ping Cui, Yupu Yang, Bowen Sun, et al. 2021. Apan: Asynchronous propagation attention network for real-time temporal graph embedding. In *Proceedings of the 2021 international conference on management of data*. 2628–2638.

[60] Renchi Yang, Jieming Shi, Keke Huang, and Xiaokui Xiao. 2022. Scalable and Effective Bipartite Network Embedding. In *Proceedings of the 2022 International Conference on Management of Data*. 1977–1991.

[61] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, and Sourav S Bhowmick. 2019. Homogeneous network embedding for massive graphs via reweighted personalized pagerank. *arXiv preprint arXiv:1906.06826* (2019).

[62] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, Juncheng Liu, and Sourav S Bhowmick. 2020. Scaling attributed network embedding to massive graphs. *arXiv preprint arXiv:2009.00826* (2020).

[63] Chuan-Yang Yin, Wen-Xu Wang, Guanrong Chen, and Bing-Hong Wang. 2006. Decoupling process for better synchronizability on scale-free networks. *Physical Review E* 74, 4 (2006), 047102.

[64] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 974–983.

[65] Hongyuan Zha and Horst D. Simon. 1999. On Updating Problems in Latent Semantic Indexing. *SIAM Journal on Scientific Computing* 21, 2 (1999), 782–791. https://doi.org/10.1137/S1064827597329266

[66] Guo-Qing Zhang, Di Wang, and Guo-Jie Li. 2007. Enhancing the transmission efficiency by edge deletion in scale-free networks. *Physical Review E* 76, 1 (2007), 017101.

[67] 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.

[68] Wentao Zhang, Yu Shen, Yang Li, Lei Chen, Zhi Yang, and Bin Cui. 2021. Alg: Fast and accurate active learning framework for graph convolutional networks. In *Proceedings of the 2021 International Conference on Management of Data*. 2366–2374.

[69] Yuyu Zhang, Liang Pang, Lei Shi, and Bin Wang. 2015. Large Scale Purchase Prediction with Historical User Actions on B2C Online Retail Platform. arXiv:1408.6515 [cs.LG]

[70] Tongya Zheng, Xinchao Wang, Zunlei Feng, Jie Song, Yunzhi Hao, Mingli Song, Xingen Wang, Xinyu Wang, and Chun Chen. 2023. Temporal Aggregation and Propagation Graph Neural Networks for Dynamic Representation. *IEEE Transactions on Knowledge and Data Engineering* (2023).

[71] Wenliang Zhong, Rong Jin, Cheng Yang, Xiaowei Yan, Qi Zhang, and Qiang Li. 2015. Stock constrained recommendation in tmall. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2287–2296.

[72] Xun Zhou, Jing He, Guangyan Huang, and Yanchun Zhang. 2015. SVD-based incremental approaches for recommender systems. *J. Comput. System Sci.* 81, 4 (2015), 717–733.

Cover Note for VLDB 2025 Submission

We thank the reviewers for the insightful comments and for offering us the 'Accept with Shepherding' option. We have considerably revised the paper to address the concerns raised, and have highlighted the major changes in the paper in blue. In the following, we provide our responses to the reviewers' comments.

META-REVIEWER

Meta-Reviewer's Summary: *All reviewers agree that the paper deals with an important problem and provides an efficient solution. The paper is, in general, well-written, and the experiments are comprehensive. There are, however, some points that need to be addressed before the paper is deemed ready for publication. Please see below for a summary of the required revisions and refer to the original reviews for details.*

Response: Thank you for your appreciation. We have carefully revised our paper based on your suggestions and have listed our responses to each revision item below.

Meta-Reviewer, Revision item R1: *Presentation needs some improvements. Please check carefully for spelling and grammatical errors. Also, some mathematical formulations are confusing; provide some intuition in plain English.*

Response: Thank you for the comment. In the revised paper, we have thoroughly corrected the spelling and grammatical errors. Additionally, we have added more explanations to help clarify our proof.

Meta-Reviewer, Revision item R2: *Discuss the limitations of your method and discuss how they would affect real-life applications.*

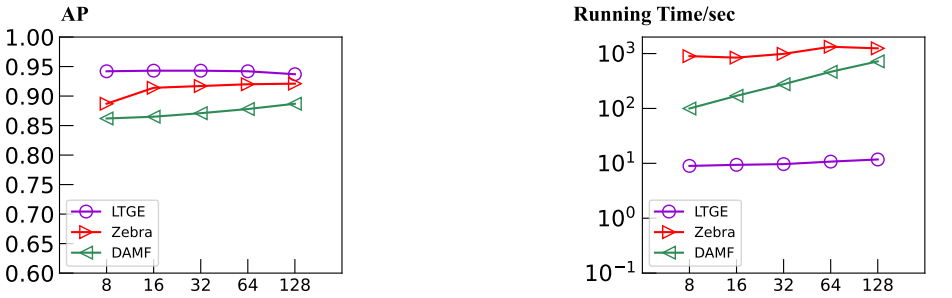
Response: Thank you for your comment. In general, our method does not consider node features in current work, which may further enhance the embedding quality as evidenced by previous T-GNNs like Zebra. A real-world application is attributed user-item interaction systems, where the node features is sometimes available for enhancing the embedding quality. For such attributed graphs, a naive solution is to fuse the feature information by concatenating the node features with the embedding generated by LTGE. In our future work, we aim to expand our methods to attributed graphs by incorporating the type of nodes and edges into the embeddings.

In the revised paper, we have added the discussions of LTGE and LTGEInc in details in Section 8. Some naive solutions to handle these issues are also provided in Section 8, and we reserve more comprehensive solutions for future work.

Meta-Reviewer, Revision item R3: *Discuss the selection of parameters; e.g., why do you use 32-dim embedding?*

Response: We appreciate the reviewers' insightful comments. For the parameters d, 32 or lower dimensions is a common choose to validate the effectiveness in previous work [5, 23, 24, 34]. While larger embedding dimensions can potentially enhance quality, they also require more computational time, making the selection a balance between effectiveness and efficiency. For most graph embedding methods, dimensions larger than 32 do not significantly improve performance but do result in considerable increases in computational time. Our work focuses on embedding generation for large-scale graphs. Consequently, in alignment with previous research, we selected an embedding dimension of 32 for testing.

To verify whether LTGE and baselines would perform better in larger dimensions, we also conducted parameters sensitive analysis on AskUbuntu dataset. The results shown in Figure I indicate that the overall conclusions of our experiments remain unchanged. We observe that LTGE experiences a slight performance drop with dimensions larger than 32. Zebra has a slight improvement beyond 32 dimensions, yet its running time increases. While DAMF benefits from higher embedding dimensions, this comes at the cost of significantly increased running time. Notably, LTGE achieves optimal performance and efficiency with each dimension, yet it requires smaller dimensions to achieve superior results. To balance the effectiveness and the efficiency, we selected 32 dimensions for our experiments. Due to the page limit, we will put the results and analysis into our technical report. In the revised paper, we have added the above discussion about the selection of dimensions.



(a) AP scores with different dimensions.

(b) Running time with different dimensions.

Figure I: Parameters Sensitive Analysis on AskUbuntu.

Meta-Reviewer, Revision item R4: *Explain better why your approach is better than the competitors that appear in Tables 4 and 6.*

Response: Thanks for the suggestion. First, LTGE outperforms all baselines in effectiveness. This is due to the fact that we properly define a similarity TPASim that preserves temporal information, and the embedding we end up with is equivalent to factoring this similarity matrix. TPASim takes into account the association between each node and different snapshots, thus being able to preserve both timing information and structural information. At the same time, LTGE significantly outperforms the existing temporal graph embedding methods in the efficiency of embedding generation. As shown in Figure 4, compared with Zebra, SOTA of temporal graph embedding methods, LTGE shows one to two orders of magnitude efficiency improvement on multiple datasets. The reason behind this is that LTGE avoids directly factorizing a dense matrix with n^2 elements, and instead factorizes a temporal matrix with at most m nonzero elements via RandomizedSVD.

In the revised paper, we have added the above explanation and analysis in Section 7. Thank you again for your nice comment.

REVIEWER 1

Reviewer 1, Detailed Evaluation D1: *The authors should emphasize that, since they are building a bipartite graph to compute the similarity, the number of edges (m) is much lesser than n'2.*

Response: We thank the reviewer for the insightful comment. In the revised paper, we have added discussion about the number of edges in the temporal bipartite graph and emphasized that the number of edges in the temporal bipartite graph is much lesser than n'2 in Section 4.2

Reviewer 1, Detailed Evaluation D2: *The decision of having k snapshots of the same length T/k should be better justified, as it may lead to a ver different snapshots. Could an even edge partition work as well? may it work better?*

Response: Thanks for your suggestions. Our decision to split each snapshot by the same length is based on the collection process of most time series datasets. In the real world, most of the time series data is collected at a fixed time length, and splitting it by the same length can simulate this process naturally. Of course, we also appreciate your suggestion of splitting by edge, which we will consider in the future.

Reviewer 1, Detailed Evaluation D3: *Make it explicit that the only update considered is edge addition: not deletions or updates.*

Response: Thanks for the comment. In the revised paper, we have emphasized that the incremental update mechanism in our work is specifically designed to handle the addition of new edges and nodes in temporal graphs, while edge deletions are not considered. In the revised paper, we have talked about the limitations and provide some naive solution in Section 8.

Reviewer 1, Detailed Evaluation D4: *The theoretical guarantee proven in Lemma 5.2 should be more clearly explained, for audiences that do not follow the maths.*

Response: Following your advice, we have carefully revised the Lemma 5.2 with plain English. Thanks you for pointing out this issue.

Reviewer 1, Detailed Evaluation D5: *The experiments otherwise look well thought out and fair. My only criticism here is that I do not understand how can you vary the number of nodes while keeping the number of edges constant, and vice versa (experiments of Fig. 5.).*

Response: Thank you for your comment. Here we follow the way in previous work [60] to verify the efficiency of the proposed methods. Specifically, we use the Erdos Renyi random graph generation model [14], which will pick a graph uniformly at random out of all graphs with n nodes and m edges with a given number of nodes and edges. Therefore, we can use this random graph generation model to generate random graphs with the same number of nodes but different numbers of edges or the same number of edges but different numbers of nodes and test the efficiency of the proposed methods.

Reviewer 1, Detailed Evaluation D6: *A reference is needed for the first sentence of Section 4.2 and for the last sentence in the "Temporal Graph Embeddings" paragraph of Section 2.*

Response: Thanks for the comment. We have added those references in the revised paper.

Reviewer 1, Detailed Evaluation D7: *Please use a grammar and spell checker. There are several articles missing (e.g., "on graph" should be "on the graph", "designs deep" should be "designs a deep"), and some verbs are conjugated incorrectly (e.g., "have a weight" should be "has a weight").*

Response: Thanks for the comment. In the revised paper, we have carefully checked and fixed these errors.

REVIEWER 2

Reviewer 2, Opportunity O1 with Detailed Evaluation: *The overall presentation of the paper could be improved. The overall presentation of the paper could be enhanced. There are typos here and there and inconsistencies in notation that should be addressed.*

Response: Thanks for your comment. In the revised paper, we have carefully modified these typos and improved the presentation of proof and analysis.

Reviewer 2, Opportunity O2 with Detailed Evaluation: *Important explanations and intuitions are missing in key places. For example, the complexity analysis in Section 6 could benefit from more detailed explanations of how the time and space complexities are derived.*

Response: Thanks for your comment. First, we give the analysis of LTGE (Algorithm 1). Constructing the temporal matrix $\mathbf{W}$ needs to operate on every edge in the original temporal graph G (lines 1–8) and perform log operation on every non-zero element (line 9). Since there are m edges in G and at most $2m$ non-zero elements in $\mathbf{W}$ as per our analysis in Section 4.2, the construction of $\mathbf{W}$ requires $O(m)$ time. Moreover, by Randomized SVD, factorizing $\mathbf{W} \in \mathbb{R}^{n\times k}$ with $O(m)$ non-zero elements to $\mathbf{U}_d \in \mathbb{R}^{n\times d}$, $\Sigma_d \in \mathbb{R}^{d\times d}$ and $\mathbf{V}_d \in \mathbb{R}^{n\times d}$ takes $O(md\log(k) + nd^2\log(k))$ [42] (lines 10). Finally, calculating $\mathbf{Z} = \mathbf{U}_d\Sigma_d \in \mathbb{R}^{n\times d}$ takes $O(nd^2)$ time (lines 11). Putting it together, the total time complexity of LTGE is $O(md\log(k) + nd^2\log(k))$. Meanwhile, it takes $O(nd)$ to store $\mathbf{Z}$, $\mathbf{U}_d$, Σ_d and $\mathbf{V}_d$, while storing $\mathbf{W}$ with $O(m)$ non-zero elements takes $O(m)$ space via compressed sparse row [4]. Thus, the space complexity of LTGE is $O(m + nd)$.

Next, we analyze the IncrementalSVD algorithm (Algorithm 2). For $\mathbf{U}_1\Sigma_1, \mathbf{V}_1, \mathbf{U}_2\Sigma_2, \mathbf{V}_2 \in \mathbb{R}^{n\times d}$, performing QR factorization takes $O(nd^2)$ time [56] (lines 1–2). For $\mathbf{R}_1, \mathbf{R}_2 \in \mathbb{R}^{d\times d}$, performing SVD of $\mathbf{R}_1\mathbf{R}_2$ takes $O(d^3)$ time (lines 3). Finally, for $\mathbf{Q}_1, \mathbf{Q}_2 \in \mathbb{R}^{n\times d}$ and $\mathbf{U}_R, \mathbf{V}_R \in \mathbb{R}^{d\times d}$, calculating $\mathbf{U}_I = \mathbf{Q}_1\mathbf{U}_R$ and $\mathbf{V}_I^\top = \mathbf{V}_R^\top\mathbf{Q}_2^\top$ takes $O(nd^2)$ time (line 4). Therefore, the time complexity of Algorithm 2 is $O(nd^2)$. Meanwhile, its space complexity is $O(nd)$.

Now, we derive the complexity of LTGEInc (Algorithm 3) when the newly added data span β snapshots. Constructing $\mathbf{W}_i$ with at most Δm non-zero elements needs $O(\Delta m)$ time (lines 1–10). Performing truncated SVD on $\mathbf{W}_i \in \mathbb{R}^{n\times\beta}$ takes $O(nd\beta)$ time [53] (line 11). Invoking Algorithm 2 takes $O(nd^2)$ time (line 12). For $\mathbf{U}_i \in \mathbb{R}^{n\times d}$ and $\Sigma_i \in \mathbb{R}^{d\times d}$, calculating $\mathbf{Z}_i = \mathbf{U}_i\Sigma_i$ takes $O(nd^2)$ time (lines 13). Therefore, the total time complexity of LTGEInc is $O(nd\beta + nd^2 + \Delta m)$. Meanwhile, LTGEInc takes $O(nd)$ to store matrices except for $\mathbf{W}_i$ which requires $O(\Delta m)$ space. Therefore, the space complexity of LTGEInc is $O(nd + \Delta m)$.

In the revised paper, we have modified the analysis of time complexity and space complexity in Section 6. Thanks again for your insight comment.

Reviewer 2, Opportunity O3 with Detailed Evaluation: *Approach's limitations should be explored better. The paper could provide more insight into potential limitations or scenarios where LTGE might not perform as well. For example, a more detailed comparison with the most recent competitor, Zebra, would strengthen the paper's positioning in the current state of the art. Similarly, the paper could benefit from a more in-depth discussion of the practical implications and potential real-world applications of LTGE.*

Response: We appreciate the reviewers' insightful comment. For LTGE, the main limitation is that it does not consider graphs with node features particularly, while T-GNNs such as Zebra have the specific module to utilize node features to enhance the quality of the embeddings. For this type of attributed graph, a naive solution is to fuse the feature information by concatenating the node features with the embedding generated by LTGE to obtain a new embedding. We will try to improve LTGE to solve the embedding generation problem in the attributed temporal graph in our future work. For LTGEInc, the main limitation is that it does not consider how to handle edge and node deletion with provable guarantees. Although some existing work [13, 72] on incremental SVD for matrix deletion can be used to solve this problem, it does not have a provable guarantee, making LTGEInc less effective in cases where nodes or edges need to be deleted frequently. In the future, we will focus on designing theoretically guaranteed algorithms for scenarios involving node and edge deletion. In the revised paper, we have added the

analysis and provide some naive solution or combination with existing works [13, 72] to mitigate these issues in Section 8.

Reviewer 2, Opportunity O4 with Detailed Evaluation: *Additional guidance on practical use cases is needed. The paper could provide additional guidance on parameter selection for different types of temporal graphs, as this is crucial for practical implementation.*

Response: Thanks for the comment. For parameters k and ρ, we conduct parameter analysis in our paper. The selection of ρ is determined by the downstream tasks and whether it is a bipartite temporal graph. We suggest choosing $\rho = 10^{-3}$ for future link prediction and top-K recommendation on non-bipartite temporal graphs while $\rho = 10^{-3}$ is also suitable for future link prediction but $\rho = 1$ should be used on top-K recommendation for bipartite temporal graphs. The selection of parameter k will not make appreciable impact for performance in this reasonable range. It is reasonable to select k as 200 to small datasets and increase it for large datasets.

For parameters d, which is the dimensions of the embedding, we have added the parameter analysis in our technical report [] due to the page limit. The results are shown in Figure I. In general, LTGE experiences a slight performance drop with dimensions larger than 32. To balance the effectiveness and efficiency, we selected 32 dimensions for our experiments.

REVIEWER 3

Reviewer 3, Opportunity O1 with Detailed Evaluation: *The usecases considered are only future prediction based. It is necessary to also see the accuracy in current snapshot tasks. The usecases considered in the experiments are only future prediction based. What is the accuracy in current snapshot tasks? For e.g., what is the accuracy for the collaboration problem as illustrated by the example in Figure 1?*

Response: Thanks for your comment. First, we would like to clarify that since our proposed methods and other temporal graph embedding methods mainly focuses on temporal graphs, so the accuracy in current snapshot is not the main task we focus on. Second, to verify the accuracy in current snapshot, we conduct static link prediction follow previous static graph embedding methods [40, 60, 61] on AskUbuntu. Specifically, we randomly shuffled all the edges without considering the generation time of the edges and selected 30% as the test set and 70% as the training set. Since Zebra and other T-GNNs do not support this type of downstream task, we choose DAMF as the competitor. The result is shown in Table 7. Although LTGE is not specifically tailored to the tasks in the current snapshot, it still outperforms baseline. Due to the page limit, we will add these results in our technical report.

Table 7: Performance of traditional link prediction on AskUbuntu.

Methods	AUC	AP
DAMF	0.915	0.925
LTGE	0.940	0.954

Reviewer 3, Opportunity O2 with Detailed Evaluation: *The improvements over prior embedding techniques seem only little while the others also generalize more (Tables 4 and 6 for e.g.) and take less time to train. Thus, the exact strong point of the proposed approach is not clear. When exactly does the proposed embedding beat the others significantly?*

Response: We appreciate the reviewers' insightful comment. In general, LTGE achieves both effectiveness and efficiency by appropriately defining similarity measures and employing sparse matrix factorization on temporal graphs. LTGE significantly outperforms large-scale static graph embedding methods such as ProNE and LocalAction, which are close to LTGE in efficiency (e.g., On Bitcoin dataset, LTGE achieves 25.6% higher AUC scores than ProNE and 34.2% higher AUC scores than LocalAction on Future Link Prediction tasks). For Zebra, which performs close to LTGE, is one to two orders of magnitude slower than LTGE in efficiency. We also design LTGEInc to enable LTGE to incremental update the embedding with theoretical error bound, and the efficiency of embedding update is also the highest in our experiments. We have added the explanation in Section 7 in the revised paper.

Reviewer 3, Opportunity O3 with Detailed Evaluation: *The experiments have been conducted with a 1TB RAM system. This is a very high requirement and higher than average for any system. It is unclear if the proposed approach can work with lower RAM. It will be interesting if the proposed approach can work with lower RAM. A study showing the maximum RAM usage by each technique will be useful to understand the efficiency of the optimizations and the strength of the contribution.*

Table 8: RAM usage on Taobao.

Methods	DAMF	LTGE
Whole RAM usage	360 GB	363 GB

Response: Thanks for the comment. In fact, LTGE and LTGEInc do not need so large RAM, the 1TB RAM system is just the environment that we use to conduct the experiments but not the requirement for our proposed methods to run without out of memory. For T-GNNs, since they use GPU to train their model and are hard to scale to large-scale graphs, it has no necessary to record the RAM usage. In Table 8, we record the largest RAM usage of LTGE and

DAMF on the largest dataset Taobao with parameter $d = 32$ and parameter $k = 1600$. The results show that LTGE needs about 363GB RAM on the temporal graphs with over one billion edges. At the same time, the current RAM usage leaves considerable room for optimization. The existing code is not fully optimized for memory efficiency. For instance, we currently use Python's list to store edges for training, which consumes more than 200GB of RAM. However, LTGE could builds the sparse temporal matrix while reading data, significantly reducing RAM overhead. To sum up, our method is still well-suited for large-scale graphs in terms of space occupancy.