Time-aware Random Walk Diffusion to Improve Dynamic Graph Learning

Jong-whi Lee and Jinhong Jung

Department of Computer Science and Engineering, Jeonbuk National University, South Korea
jong.whi.lee@jbnu.ac.kr, jinhongjung@jbnu.ac.kr

Abstract

How can we augment a dynamic graph for improving the performance of dynamic graph neural networks? Graph augmentation has been widely utilized to boost the learning performance of GNN-based models. However, most existing approaches only enhance spatial structure within an input static graph by transforming the graph, and do not consider dynamics caused by time such as temporal locality, i.e., recent edges are more influential than earlier ones, which remains challenging for dynamic graph augmentation. In this work, we propose TIARA (Time-aware Random Walk Diffusion), a novel diffusion-based method for augmenting a dynamic graph represented as a discrete-time sequence of graph snapshots. For this purpose, we first design a time-aware random walk proximity so that a surfer can walk along the time dimension as well as edges, resulting in spatially and temporally localized scores. We then derive our diffusion matrices based on the time-aware random walk, and show they become enhanced adjacency matrices that both spatial and temporal localities are augmented. Throughout extensive experiments, we demonstrate that TIARA effectively augments a given dynamic graph, and leads to significant improvements in dynamic GNN models for various graph datasets and tasks.

Introduction

Dynamic graphs represent various real-world relationships that dynamically occur over time. Learning such dynamic graphs has recently attracted considerable attention from machine learning communities (Skarding, Gabrys, and Musial 2021; Han et al. 2021), and plays a crucial role in diverse applications such as link prediction (Yang et al. 2021; Pareja et al. 2020), node or edge classification (Xu et al. 2019; Pareja et al. 2020), time-series traffic forecasting (Wu et al. 2020; Guo et al. 2019), knowledge completion (Jung, Jung, and Kang 2021), and pandemic forecasting (Panagopoulos, Nikolentzos, and Vazirgiannis 2021). Over the last years, many researchers have put tremendous effort into developing interesting methods by sophisticatedly fusing GNNs and recurrent neural networks (RNN) or attention mechanisms for continuous-time (Xu et al. 2020; Rossi et al. 2020) and discrete-time (Seo et al. 2018; Pareja et al. 2020; Yang et al. 2021) dynamic graphs.

With the astonishing progress of GNNs, diverse augmentation techniques (Zhao et al. 2022; Yoo, Shim, and Kang 2022) have been proposed to increase the generalization power of GNN models, especially on a static graph. Previous approaches mainly transform the topological structure of the input graph. For example, drop-based methods stochastically remove a certain number of edges (Rong et al. 2020) or nodes (Feng et al. 2020) at each training epoch in a similar manner to dropout regularization. On the contrary, diffusion methods (Klicpera, Weißenberger, and Günnemann 2019) insert additional edges having weights scored by graph diffusions such as Personalized PageRank (Tong, Faloutsos, and Pan 2006), thereby augmenting a spatial locality around each node and improving graph convolution.

However, the aforementioned techniques assume to augment data within a static graph, and dynamic graph augmentation problem has not yet been comprehensively studied. Unlikely static graphs, dynamic graphs change or evolve over time by their nature; thus, dynamic graph augmentation needs to simultaneously consider temporal dynamics as well as spatial structure. More specifically, as verified in previous works (Rossi et al. 2020; Shin 2017; Lee, Shin, and Faloutsos 2020), real-world dynamic graphs exhibit *temporal locality* indicating that graph objects such as nodes and triangles tend to be more affected by more recent edges than older ones, i.e., edges closer to a specific object in time are more likely to provide important information. Naively applying a static augmentation method to each time step cannot consider such a temporal locality.

In this work, we propose TIARA (Time-aware Random Walk Diffusion), a novel diffusion-based augmentation method for a discrete-time dynamic graph which is represented by a temporal sequence of graph snapshots. TIARA aims to augment both spatial and temporal localities of each graph snapshot. For this purpose, we design a time-aware random walk that a surfer randomly moves around nodes or a time-axis to measure spatially and temporally localized scores. We then derive time-aware random walk diffusion from the scores, and interpret it as the combination of spatial and temporal augmenters. Our diffusion matrices are used as augmented adjacency matrices for any dynamic GNN models in discrete-time domain. We further adopt approximate techniques such as power iteration and sparsification to reduce a heavy cost for computing the diffusion matrices.

Our contributions are summarized as follows:

- **Method.** We propose TIARA, a novel and model-agnostic method for dynamic graph augmentation using time-aware random walks. TiaRa strengthens not only a spatial locality but also a temporal locality of a dynamic graph so that dynamic GNNs perform better.

- **Analysis.** We analyze how TiaRa augments both spatial and temporal localities (Theorem 1) and complexities of TiaRa (Theorem 2) in real dynamic graphs.
- **Experiments.** We demonstrate that TiaRa effectively augments a given dynamic graph, and leads to consistent improvements in GNNs for temporal link prediction and node classification tasks.

The code of TiaRa and the datasets are publicly available at **https://github.com/dev-jwel/TiaRa**.

Related Work

Augmentation for Static GNNs. Graph augmentation (Zhao et al. 2022) aims to reduce over-fitting for training GNN models by modifying an input graph. DropEdge (Rong et al. 2020) or DropNode (Feng et al. 2020) randomly drop edges or nodes at each epoch. These augment the diversity of the input graph by creating different copies sampled from the graph. GDC (Klicpera, Weißenberger, and Günnemann 2019) adds new edges weighted by a graph diffusion derived from node proximities. GDC boosts a spatial locality of the graph so that a GNN can consider adjacent nodes as well as distant ones during their convolutions, enhancing its representation power. However, most of existing methods are limited to augment dynamic graphs because they do not consider temporal properties.

GNNs and Augmentation for Dynamic Graphs. Dynamic graphs (Kazemi et al. 2020) are categorized as: discrete-time dynamic graphs (DTDG) and continuous-time dynamic graphs (CTDG) where a DTDG is represented as a sequence of graph snapshots with multiple discrete time steps while a CTDG is represented as a set of temporal edges whose time-stamps have continuous values. It is straightforward to convert a CTDG to a DTDG by distributing the continuous-time edges into multiple bins in chronological order, but the reverse is not possible because continuous-time values are generally lacked in most DTDGs (Yang et al. 2021), i.e., models for DTDGs can be applied to CTDGs, but the reverse is rather limited. Hence, we narrow our focus to representation learning on DTDGs.

Dynamic GNNs have rapidly advanced under the framework that closely integrates GNNs and temporal sequence models such as RNNs to capture spatial and temporal relations on dynamic graphs (Skarding, Gabrys, and Musial 2021). GCRN (Seo et al. 2018) uses a GCN to produce node embeddings on each graph snapshot, and then forwards them to an LSTM for modeling temporal dynamics. STAR (Xu et al. 2019) utilizes a GRU combined with spatial and temporal attentions. DySat (Sankar et al. 2020) employs a self-attention strategy to aggregate spatial neighborhood and temporal dynamics. EvolveGCN (Pareja et al. 2020) evolves the parameters of GCNs using RNNs. To consider hierarchical properties in real graphs, HTGN (Yang et al. 2021) extends the framework to hyperbolic space.

As a related method, MeTA (Wang et al. 2021) adaptively augments a temporal graph based on predictions of a temporal graph network, which perturbs time and removes or adds edges. However, it is difficult to employ MeTA for the aforementioned DTDG models because MeTA is designed for CTDGs requiring continuous-time values.

Preliminaries

Random Walk with Restart (RWR). Our work is related to RWR which measures node similarity scores that are spatially localized to seed node s (Nassar, Kloster, and Gleich 2015), i.e., scores of nearby nodes highly associated with s are high while those of distant nodes are low. Diffusion methods such as GDC exploit RWR to augment a spatial locality.

Let $\mathbf{x}_s$ be a vector of RWR scores w.r.t. the seed node s. Given a row-normalized adjacency matrix $\tilde{\boldsymbol{\mathcal{A}}}$ and a restart probability α, the vector $\mathbf{x}_s$ is represented as follows:

$$\mathbf{x}_s = (1-\alpha)\tilde{\boldsymbol{\mathcal{A}}}^{\top}\mathbf{x}_s + \alpha\mathbf{i}_s \Leftrightarrow \mathbf{x}_s = \alpha\mathbf{L}^{-1}\mathbf{i}_s \Leftrightarrow \mathbf{x}_s = \boldsymbol{\mathcal{L}}^{\text{rwr}}\mathbf{i}_s$$

where $\mathbf{L} = \boldsymbol{\mathcal{I}}_n - (1-\alpha)\tilde{\boldsymbol{\mathcal{A}}}^{\top}$ is the random-walk normalized Laplacian matrix, and $\mathbf{i}_s$ is the s-th unit vector. Notice that $\boldsymbol{\mathcal{L}}^{\text{rwr}} = \alpha\mathbf{L}^{-1}$ is a column-stochastic transition matrix interpreted as a diffusion kernel that diffuses a given distribution such as $\mathbf{i}_s$ on the graph through RWR.

Problem Formulation. A discrete-time dynamic graph (DTDG) $\mathcal{G}$ is represented as a sequence $\{\mathcal{G}_1, \cdots, \mathcal{G}_T\}$ of snapshots in a chronological order where T is the number of time steps (Skarding, Gabrys, and Musial 2021). Each snapshot $\mathcal{G}_t = (\mathcal{V}, \mathcal{E}_t, \mathbf{F}_t)$ is a graph with a shared set $\mathcal{V}$ of nodes and a set $\mathcal{E}_t$ of edges at time t where $n = |\mathcal{V}|$ is the number of nodes. $\mathbf{F}_t \in \mathbb{R}^{n\times d}$ is an initial node feature matrix where d is a feature dimension, and $\mathbf{A}_t \in \mathbb{R}^{n\times n}$ denotes the sparse and self-looped adjacency matrix of $\mathcal{G}_t$. The node representation learning on the dynamic graph $\mathcal{G}$ aims to learn a function $\mathcal{F}_\Theta(\cdot)$ parameterized by Θ and produce hidden node embeddings $\mathbf{H}_t \in \mathbb{R}^{n\times d}$ for each time t, represented as:

$$\mathbf{H}_t = \mathcal{F}_\Theta(\tilde{\boldsymbol{\mathcal{A}}}_t, \mathbf{F}_t, \mathbf{H}_{t-1}) \tag{1}$$

where $\tilde{\boldsymbol{\mathcal{A}}}_t$ is a normalized adjacency matrix of $\mathbf{A}_t$, and $\mathbf{H}_{t-1}$ contains the latest hidden embeddings before time t. The above framework of Equation (1) is generally adopted in existing methods (Seo et al. 2018; Pareja et al. 2020; Yang et al. 2021) for learning DTDGs where $\mathcal{F}_\Theta(\cdot)$ is usually designed by the combination of GNNs and RNNs.

Problem 1 (Dynamic Graph Augmentation). *Given a temporal sequence $\{\mathbf{A}_1, \cdots, \mathbf{A}_T\}$ of $\mathcal{G}$, the problem is to generate a sequence of new adjacency matrices improving the performance of a model $\mathcal{F}_\Theta(\cdot)$.* □

Proposed Method

We depict the overall framework of TiaRa in Figure 1. Given $\{\mathbf{A}_1, \cdots, \mathbf{A}_T\}$ of a dynamic graph $\mathcal{G}$, our TiaRa aims to produce a time-aware random walk diffusion matrix $\tilde{\boldsymbol{\mathcal{X}}}_t \in \mathbb{R}^{n\times n}$ for each time step t using two diffusion based modules, called *spatial* and *temporal* augmenters.

The spatial augmenter enhances a spatial locality of $\mathbf{A}_t$ using random walks, resulting in a spatial diffusion matrix $\boldsymbol{\mathcal{S}}_t$. The temporal augmenter receives the previous $\tilde{\boldsymbol{\mathcal{X}}}_{t-1}$ that

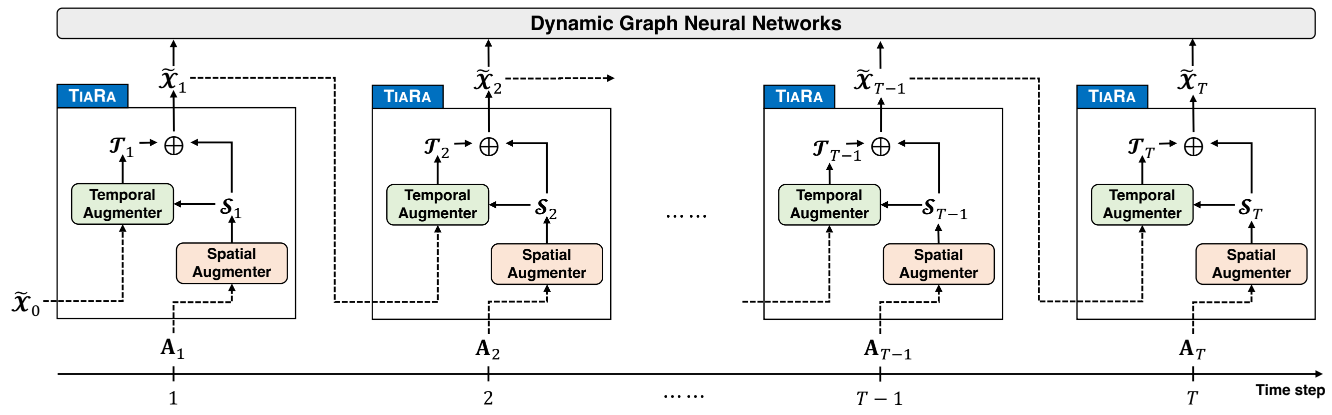


Figure 1: Overall architecture of TiaRa. Given the adjacency matrix $\mathbf{A}_t$ at time t, TiaRa outputs a time-aware random walk diffusion matrix $\tilde{\mathcal{X}}_t$ combined with spatial augmenter $\mathcal{S}_t$ and temporal augmenter $\mathcal{T}_t$ after sparsification.

contains information squashed from the initial time to $t-1$, and then disseminates it through $\mathcal{S}_t$ at the current t. This leads to a temporal diffusion matrix $\mathcal{T}_t$ in which a temporal locality is magnified. Finally, TiaRa linearly combines $\mathcal{S}_t$ and $\mathcal{T}_t$, and sparsifies to form $\tilde{\mathcal{X}}_t$. We replace each adjacency matrix $\mathbf{A}_t$ with $\tilde{\mathcal{X}}_t$ for the inputs of dynamic GNN models. If necessary, we simply use edges of the graph represented by $\tilde{\mathcal{X}}_t$ without weights, or make the graph undirected by using $(\tilde{\mathcal{X}}_t + \tilde{\mathcal{X}}_t^\top)/2$ after the sparsification.

Time-aware Random Walk with Restart

It is limited to directly employ RWR in a dynamic graph because RWR measures only spatially localized scores in a single static graph. In this section, we extend RWR to Time-aware RWR (TRWR) so that TRWR produces node-to-node scores which are spatially and temporally localized.

One idea for TRWR is to virtually connect identical nodes from $\mathcal{G}_t$ to $\mathcal{G}_{t+1}$ for each time step t (Huang, Sun, and Wang 2021) as shown in Figure 2. Then, a random surfer not only moves around the current $\mathcal{G}_t$ but also jumps to the next $\mathcal{G}_{t+1}$; thus, the surfer becomes time-aware. In the beginning, the surfer starts from a seed node s at the initial time step (e.g., $t=1$). After a few movements, suppose the surfer is at node u in $\mathcal{G}_t$. Then, it takes one of the following actions:

- **Action 1) Random walk.** The surfer randomly moves to one of the neighbors from node u in the current graph $\mathcal{G}_t$ with probability $1-\alpha-\beta$.
- **Action 2) Restart.** The surfer goes back to the seed node s in $\mathcal{G}_t$ with probability α.
- **Action 3) Time travel.** The surfer does time travel from node u in $\mathcal{G}_t$ to that node in $\mathcal{G}_{t+1}$ with probability β.

where α and β are called restart and time travel probabilities, respectively, and $0 < \alpha + \beta < 1$. Note that we do not allow the surfer to move backward from $\mathcal{G}_{t+1}$ to $\mathcal{G}_t$ because the future information at time $t+1$ should be prevented when we make a prediction at time t.

Through TRWR, the vector $\mathbf{x}_t \in \mathbb{R}^n$ of stationary probabilities that the surfer visits each node from the seed node s

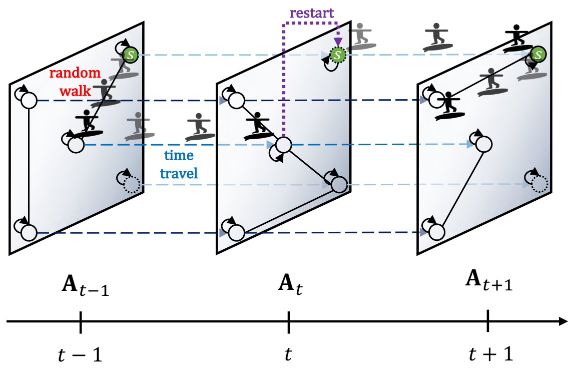


Figure 2: Illustration of how a time-aware random surfer moves around a dynamic graph where dashed arrows indicate virtually connected edges along the time-axis.

in $\mathcal{G}_t$ is recursively represented as follows:

$$\mathbf{x}_{t,s} = \underbrace{(1-\alpha-\beta)\tilde{\mathcal{A}}_t^\top \mathbf{x}_{t,s}}_{\text{Random walk}} + \underbrace{\alpha \mathbf{i}_s}_{\text{Restart}} + \underbrace{\beta \mathbf{x}_{t-1,s}}_{\text{Time travel}} \quad (2)$$

where $\mathbf{i}_s$ is the s-th unit vector of size n. $\tilde{\mathcal{A}}_t$ is a row-normalized matrix of $\mathbf{A}_t$ (i.e., $\tilde{\mathcal{A}}_t = \mathbf{D}_t^{-1}\mathbf{A}_t$ where $\mathbf{A}_t$ is a self-looped adjacency matrix and $\mathbf{D}_t$ is a diagonal out-degree matrix of $\mathbf{A}_t$). If $t=0$, we define $\mathbf{x}_{0,s}$ as $\mathbf{i}_s$.

In the above equation, the random walk part propagates scores of $\mathbf{x}_{t,s}$ over $\tilde{\mathcal{A}}_t$. The restart part makes the scores spatially localized around the seed node s, which is controlled by α. The time travel part injects scores of the previous $\mathbf{x}_{t-1,s}$ to make $\mathbf{x}_{t,s}$ temporally localized, which is controlled by β. Notice TRWR extends RWR to a discrete-time dynamic graph, i.e., $\beta = 0$ leads to RWR scores on each graph snapshot without considering temporal information.

Time-aware Random Walk Diffusion Matrices

In Equation (2), $\mathbf{x}_{t,s} \in \mathbb{R}^{n\times 1}$ is a column vector of a probability distribution w.r.t. a seed node s. For all seeds $s \in \mathcal{V}$, we horizontally stack $\{\mathbf{x}_{t,s}\}$ to form $\mathcal{X}_t \in \mathbb{R}^{n\times n}$ such that $\mathbf{x}_{t,s}$ is the s-th column of $\mathcal{X}_t$, i.e., $\mathbf{x}_{t,s} = \mathcal{X}_t \mathbf{i}_s$. We call $\mathcal{X}_t$

a time-aware random walk diffusion matrix at time t. The derivation of $\boldsymbol{\mathcal{X}}_t$ starts by moving the term of the random walk to the left side in Equation (2) as follows:

$$\left(\boldsymbol{\mathcal{I}}_n - (1-\alpha-\beta)\tilde{\boldsymbol{\mathcal{A}}}_t^\top\right)\mathbf{x}_{t,s} = \alpha\mathbf{i}_s + \beta\mathbf{x}_{t-1,s}$$

Let $\mathbf{L}_t \coloneqq \boldsymbol{\mathcal{I}}_n - (1-\alpha-\beta)\tilde{\boldsymbol{\mathcal{A}}}_t^\top$ where $\boldsymbol{\mathcal{I}}_n$ is an $n \times n$ identity matrix, and $\mathbf{x}_{t-1,s} = \boldsymbol{\mathcal{X}}_{t-1}\mathbf{i}_s$ as described above. Thus, $\mathbf{x}_{t,s}$ is written as the following:

$$\begin{aligned}\mathbf{x}_{t,s} &= \mathbf{L}_t^{-1}\left(\alpha\mathbf{i}_s + \beta\boldsymbol{\mathcal{X}}_{t-1}\mathbf{i}_s\right)\\ &= \left(\alpha\mathbf{L}_t^{-1}\boldsymbol{\mathcal{I}}_n + \beta\mathbf{L}_t^{-1}\boldsymbol{\mathcal{X}}_{t-1}\right)\mathbf{i}_s = \boldsymbol{\mathcal{X}}_t\mathbf{i}_s \qquad (3)\end{aligned}$$

where $\boldsymbol{\mathcal{X}}_t = \alpha\mathbf{L}_t^{-1}\boldsymbol{\mathcal{I}}_n + \beta\mathbf{L}_t^{-1}\boldsymbol{\mathcal{X}}_{t-1}$ for $t > 0$, and $\boldsymbol{\mathcal{X}}_0 = \boldsymbol{\mathcal{I}}_n$ because $\mathbf{x}_{0,s}$ is defined as $\mathbf{i}_s$.

Spatial and Temporal Augmenters. We obtain the recurrence relation of $\boldsymbol{\mathcal{X}}_t$ from Equation (3), and further rearrange it to interpret the process as follows:

$$\begin{aligned}\boldsymbol{\mathcal{X}}_t &= \alpha\mathbf{L}_t^{-1}\boldsymbol{\mathcal{I}}_n + \beta\mathbf{L}_t^{-1}\boldsymbol{\mathcal{X}}_{t-1}\\ &= \frac{\alpha}{\alpha+\beta}\left[(\alpha+\beta)\mathbf{L}_t^{-1}\boldsymbol{\mathcal{I}}_n\right] + \frac{\beta}{\alpha+\beta}\left[(\alpha+\beta)\mathbf{L}_t^{-1}\boldsymbol{\mathcal{X}}_{t-1}\right]\end{aligned}$$

In the above, we set $\boldsymbol{\mathcal{L}}_t^{\text{rwr}} = (\alpha+\beta)\mathbf{L}_t^{-1}$ which is the diffusion kernel by RWR on the graph $\mathcal{G}_t$ where its restart probability is $\alpha+\beta$. Let $\gamma = \beta/(\alpha+\beta)$ where $0 < \gamma < 1$; then, $\boldsymbol{\mathcal{X}}_t$ is represented as follows:

$$\boldsymbol{\mathcal{X}}_t = (1-\gamma)\underbrace{\left(\boldsymbol{\mathcal{L}}_t^{\text{rwr}}\boldsymbol{\mathcal{I}}_n\right)}_{\boldsymbol{\mathcal{S}}_t} + \gamma\underbrace{\left(\boldsymbol{\mathcal{L}}_t^{\text{rwr}}\boldsymbol{\mathcal{X}}_{t-1}\right)}_{\boldsymbol{\mathcal{T}}_t = \boldsymbol{\mathcal{S}}_t\boldsymbol{\mathcal{X}}_{t-1}} \qquad (4)$$

where $\boldsymbol{\mathcal{S}}_t$ is a spatial diffusion matrix, and $\boldsymbol{\mathcal{T}}_t$ is a temporal diffusion matrix.

The meaning of $\boldsymbol{\mathcal{S}}_t$ is the result of diffusing the s-th column $\mathbf{i}_s$ of $\boldsymbol{\mathcal{I}}_n$ through $\boldsymbol{\mathcal{L}}_t^{\text{rwr}}$ for each node s. This is interpreted as the augmentation of a spatial locality of each node through RWR within $\mathcal{G}_t$. On the other hand, $\boldsymbol{\mathcal{T}}_t$ is the result of diffusing $\mathbf{x}_{s,t-1}$ of $\boldsymbol{\mathcal{X}}_{t-1}$ through $\boldsymbol{\mathcal{L}}_t^{\text{rwr}}$ for each node s. Note that $\mathbf{x}_{s,t-1}$ contains the probabilities that the surfer visits each node starting from node s during the travel from the initial time to $t-1$. Thus, it spreads the past proximities of $\mathbf{x}_{s,t-1}$ in the current $\mathcal{G}_t$ through $\boldsymbol{\mathcal{L}}_t^{\text{rwr}}$, which consequently reflects the temporal information to $\mathcal{G}_t$.

The final diffusion matrix $\boldsymbol{\mathcal{X}}_t$ is a convex combination between $\boldsymbol{\mathcal{S}}_t$ and $\boldsymbol{\mathcal{T}}_t$ w.r.t. γ, which is denoted by $\oplus$ in Figure 1. Notice that $\boldsymbol{\mathcal{X}}_t$ is a column stochastic transition matrix for every time step t, which is proved in Lemma 1, implying that as an augmented adjacency matrix, $\tilde{\boldsymbol{\mathcal{A}}}_t$ can be replaced with $\boldsymbol{\mathcal{X}}_t^\top$ for the input of GNNs in Equation (1).

Interpretation. We further analyze how $\boldsymbol{\mathcal{X}}_t$ reflects the spatial and temporal information of the input dynamic graph. For this purpose, we first obtain the closed-form expression of $\boldsymbol{\mathcal{X}}_t$ which is described in Theorem 1.

Theorem 1. *The closed-form expression of $\boldsymbol{\mathcal{X}}_t$ is:*

$$\boldsymbol{\mathcal{X}}_t = (1-\gamma)\left(\sum_{i=0}^{t-2}\gamma^i\boldsymbol{\mathcal{L}}^{\text{rwr}}_{t\leftsquigarrow t-i}\right) + \gamma^{t-1}\boldsymbol{\mathcal{L}}^{\text{rwr}}_{t\leftsquigarrow 1} \qquad (5)$$

where $\boldsymbol{\mathcal{L}}^{\text{rwr}}_{j\leftsquigarrow i} = \boldsymbol{\mathcal{L}}^{\text{rwr}}_j\boldsymbol{\mathcal{L}}^{\text{rwr}}_{j-1}\cdots\boldsymbol{\mathcal{L}}^{\text{rwr}}_i$ for $j > i$, and $\boldsymbol{\mathcal{L}}^{\text{rwr}}_{i\leftsquigarrow i} = \boldsymbol{\mathcal{L}}^{\text{rwr}}_i$.

Proof. It is proved by mathematical induction, and the detailed proof is described in Appendix A. □

In the theorem, $\boldsymbol{\mathcal{L}}^{\text{rwr}}_{j\leftsquigarrow i}$ indicates a random walk diffusion traveling from former time i toward latter time j. According to Equation (5), $\boldsymbol{\mathcal{X}}_t$ is concisely represented as:

$$\boldsymbol{\mathcal{X}}_t \propto \overbrace{\underbrace{\gamma^0\boldsymbol{\mathcal{L}}^{\text{rwr}}_t + \gamma^1\boldsymbol{\mathcal{L}}^{\text{rwr}}_{t\leftsquigarrow t-1} + \cdots + \gamma^{t-2}\boldsymbol{\mathcal{L}}^{\text{rwr}}_{t\leftsquigarrow 2} + \frac{\gamma^{t-1}}{1-\gamma}\boldsymbol{\mathcal{L}}^{\text{rwr}}_{t\leftsquigarrow 1}}_{\Longleftarrow\ \text{Emphasized} \qquad\qquad \text{Decayed}\ \Longrightarrow}}^{\text{Augmentation of spatial and temporal localities}}$$

Note that $\boldsymbol{\mathcal{X}}_t$ is more affected by the information close to time t than that passed from the distant past. The influence of $\boldsymbol{\mathcal{L}}^{\text{rwr}}_{t\leftsquigarrow k}$ is decayed by γ as time k is further away from time t, while it is emphasized as time k is near to time t where the ratio γ is interpreted as a *temporal decay ratio*. This explanation is consistent with the temporal locality, i.e., the tendency that recent edges are more influential than older ones. Combined with the spatial diffusion $\boldsymbol{\mathcal{L}}^{\text{rwr}}_t$, the result of $\boldsymbol{\mathcal{X}}_t$ augments both spatial and temporal localities in $\mathcal{G}_t$.

Discussion. TiaRa is a generalized version of GDC with PPR kernel to dynamic graphs since TiaRa with $\beta = 0$ spatially augments data within a single $\mathcal{G}_t$ at each time, which is exactly what GDC does. However, GDC does not consider temporal information for its augmentation, and it performs worse than TiaRa as shown in Tables 2 and 3.

Algorithm for TiaRa

Most graph diffusions involve heavy computational cost, especially for a large graph, and result in a dense matrix. The computation of $\boldsymbol{\mathcal{X}}_t$ also exhibits the same issue, and thus we adopt approximate techniques to alleviate the problem. Including the approximate strategies, the procedure of TiaRa is summarized in Algorithm 1 where $\tilde{\boldsymbol{\mathcal{X}}}_0$ is set to $\boldsymbol{\mathcal{I}}_n$.

Power iteration. The main bottleneck for obtaining $\boldsymbol{\mathcal{X}}_t$ is to compute the matrix inversion $\mathbf{L}_t^{-1}$ of $\boldsymbol{\mathcal{L}}^{\text{rwr}}_t$ in Equation (4), which requires $O(n^3)$ time. Instead of directly calculating the inversion, we use power iteration (lines $9\sim15$) based on the following (Yoon, Jung, and Kang 2018):

$$\mathbf{L}_t^{-1} = \sum_{k=0}^{\infty}c^k\left(\tilde{\boldsymbol{\mathcal{A}}}_t^\top\right)^k \approxeq \sum_{k=0}^{K}c^k\left(\tilde{\boldsymbol{\mathcal{A}}}_t^\top\right)^k$$

where $c = 1-\alpha-\beta$, and K is the number of iterations. Let $\mathbf{M}_t^{(k)}$ be the result after k iterations; then, it is recursively represented as follows:

$$\mathbf{M}_t^{(k)} = \boldsymbol{\mathcal{I}}_n + c\tilde{\boldsymbol{\mathcal{A}}}_t^\top\mathbf{M}_t^{(k-1)}$$

where $\mathbf{M}_t^{(0)} = \boldsymbol{\mathcal{I}}_n$ and $\mathbf{M}_t^{(K)} \approxeq \mathbf{L}_t^{-1}$. Note $\tilde{\boldsymbol{\mathcal{A}}}_t$ is a normalized adjacency matrix (line 1) in which self-loops are added, as traditional GNNs usually do. The approximate error is bounded by c^k, and converges to 0 as $k \to \infty$ (see Lemma 2). After that, we set $\boldsymbol{\mathcal{L}}^{\text{rwr}}_t \leftarrow (1-c)\mathbf{M}_t^{(K)}$ (line 13).

At a glance, each iteration seems to take $O(n^3)$ time for the matrix multiplication, but it is much faster than that since each snapshot $\mathcal{G}_t$ is sparse in most cases. More specifically, only a few nodes form edges at each time step in real graphs. We call such nodes *activated* where $\mathcal{V}_t$ is the set of activated nodes at time t, and $n_t = |\mathcal{V}_t|$. In each $\mathcal{G}_t$, a surfer can move only between activated nodes, i.e., only pairs of nodes in $\mathcal{V}_t$ are diffused. As seen in Table 1, the average $\bar{n}_t$ of n_t over time is smaller than n except the Brain dataset.

This allows us to do the power iteration on the sub-matrix $\tilde{\boldsymbol{\mathcal{A}}}_{\mathcal{V}_t} \in \mathbb{R}^{n_t \times n_t}$ of $\tilde{\boldsymbol{\mathcal{A}}}_t$ for nodes in $\mathcal{V}_t$ where m_t is the number of non-zeros of $\tilde{\boldsymbol{\mathcal{A}}}_{\mathcal{V}_t}$. Then, an iteration takes $O(m_t n_t)$ time for a sparse matrix multiplication. Note m_t is linearly proportional to n_t in real graphs, i.e., $m_t = C_t n_t$ where C_t is a constant. Let $\bar{m}_t$ be the average number of edges over time. As seen in Table 1, $\bar{C}_t = \bar{m}_t/\bar{n}_t$ is smaller than $\bar{n}_t$. Thus, each iteration takes $O(n_t^2)$ time in average; overall, it takes $O(n_t^2 K + nK)$ time and $O(n_t^2 + n)$ space for $\boldsymbol{\mathcal{L}}_t^{\text{rwr}}$ (as only n_t nodes are diffused). More details are provided in Appendix A.

Sparsification. Another bottleneck is that $\boldsymbol{\mathcal{X}}_t$ is likely to be dense by repeatedly multiplying $\boldsymbol{\mathcal{S}}_t \boldsymbol{\mathcal{X}}_{t-1}$ (line 4) as time t increases where $\boldsymbol{\mathcal{S}}_t = \boldsymbol{\mathcal{L}}_t^{\text{rwr}}$. This could be problematic in terms of space as well as running time, especially for graph convolutions since $\boldsymbol{\mathcal{X}}_t$ is used as an adjacency matrix. To alleviate this issue, we adopt a sparsification technique suggested in (Klicpera, Weißenberger, and Günnemann 2019). As established in Theorem 1, the graph structure of $\boldsymbol{\mathcal{X}}_t$ is spatially and temporally localized, which allows us to drop small entries of $\boldsymbol{\mathcal{X}}_t$, resulting in the sparse $\tilde{\boldsymbol{\mathcal{X}}}_t$. For this, we use a filtering threshold ϵ to set values of $\boldsymbol{\mathcal{X}}_t$ below ϵ to zero (line 6). This strategy has two advantages. First, it keeps $\tilde{\boldsymbol{\mathcal{X}}}_t$ sparse at each time. Second, it reduces the cost for processing $\boldsymbol{\mathcal{S}}_t \tilde{\boldsymbol{\mathcal{X}}}_{t-1}$ as $\boldsymbol{\mathcal{S}}_t$ and $\tilde{\boldsymbol{\mathcal{X}}}_{t-1}$ are sparse. After the sparsification, we normalize $\tilde{\boldsymbol{\mathcal{X}}}_{t-1}$ (line 7) column-wise. As shown in Figure 4, this sparsification makes the augmentation process fast and lightweight with tiny errors while it does not harm predictive accuracy too much, or can even improve.

Theorem 2 (Complexity Analysis)**.** *For each time step t,* TIARA *takes $O(n_t n/\epsilon + n_t^2 K)$ time on average, and produces $\tilde{\boldsymbol{\mathcal{X}}}_t$ consuming $O(n/\epsilon)$ space where n is the number of total nodes, n_t is the number of activated nodes at time t, K is the number of iterations, and ϵ is a filtering threshold.*

Proof. The proof is provided in Appendix A. □

Discussion. Theorem 2 implies that TIARA is faster than $O(n^3)$, and uses space less than $O(n^2)$ for storing $\tilde{\boldsymbol{\mathcal{X}}}_t$ in most real dynamic graphs. Nevertheless, its time complexity can reach $O(n^2)$ for a graph such as the Brain dataset; thus, for larger graphs, its scalability can be limited. However, TIARA is based on matrix operations which are easy-to-accelerate using GPUs, and other diffusion methods such as GDC lie at the same complexity. Furthermore, there are extensive works of efficient RWR computations (Andersen, Chung, and Lang 2006; Jung et al. 2017; Shin et al. 2015; Wang et al. 2017; Hou et al. 2021) and accelerated multiplications of sparse matrices (Srivastava et al. 2020), which can make TIARA scalable. In this work, we focus on effectively augmenting a dynamic graph, and leave further computational optimization on the augmentation as future work.

```
Algorithm 1: TIARA at time $t$
Input: adjacency matrix $\mathbf{A}_t$, previous time-aware diffusion matrix
    $\tilde{\boldsymbol{\mathcal{X}}}_{t-1}$, restart probability $\alpha$, time travel probability $\beta$, number
    $K$ of iterations, filtering threshold $\epsilon$
Output: time-aware diffusion matrix $\tilde{\boldsymbol{\mathcal{X}}}_t^\top$
 1: $\tilde{\boldsymbol{\mathcal{A}}}_t \leftarrow \mathbf{D}_t^{-1}\mathbf{A}_t$ where $\mathbf{D}_t = \text{diag}(\mathbf{A}_t\mathbf{1})$
 2: $\boldsymbol{\mathcal{L}}_t^{\text{rwr}} \leftarrow$ POWER-ITERATION($\tilde{\boldsymbol{\mathcal{A}}}_t$, $\alpha$, $\beta$, $K$)
 3: $\boldsymbol{\mathcal{S}}_t \leftarrow \boldsymbol{\mathcal{L}}_t^{\text{rwr}}$                              ▷ Spatial augmenter
 4: $\boldsymbol{\mathcal{T}}_t \leftarrow \boldsymbol{\mathcal{S}}_t \tilde{\boldsymbol{\mathcal{X}}}_{t-1}$                            ▷ Temporal augmenter
 5: $\boldsymbol{\mathcal{X}}_t \leftarrow (1-\gamma)\boldsymbol{\mathcal{S}}_t + \gamma\boldsymbol{\mathcal{T}}_t$ where $\gamma = \beta/(\alpha+\beta)$
 6: $\tilde{\boldsymbol{\mathcal{X}}}_t \leftarrow$ filter entries of $\boldsymbol{\mathcal{X}}_t$ if their weights are $< \epsilon$
 7: normalize $\tilde{\boldsymbol{\mathcal{X}}}_t$ column-wise
 8: return $\tilde{\boldsymbol{\mathcal{X}}}_t^\top$
 9: function POWER-ITERATION($\tilde{\boldsymbol{\mathcal{A}}}_t$, $\alpha$, $\beta$, $K$)
10:     set $c \leftarrow 1-\alpha-\beta$ and $\mathbf{M}_t^{(0)} \leftarrow \boldsymbol{\mathcal{I}}_n$
11:     for $k \leftarrow 1$ to $K$ do
12:         $\mathbf{M}_t^{(k)} \leftarrow \boldsymbol{\mathcal{I}}_n + c\tilde{\boldsymbol{\mathcal{A}}}_t^\top \mathbf{M}_t^{(k-1)}$
13:     $\boldsymbol{\mathcal{L}}_t^{\text{rwr}} \leftarrow (1-c)\mathbf{M}_t^{(K)}$ where $\mathbf{M}_t^{(K)} \approxeq \mathbf{L}_t^{-1}$
14:     normalize $\boldsymbol{\mathcal{L}}_t^{\text{rwr}}$ column-wise and return $\boldsymbol{\mathcal{L}}_t^{\text{rwr}}$
15: end function
```

Table 1: Summary of datasets. n and m are the total numbers of nodes and edges, resp. T and L are the numbers of time steps and labels, resp. $\bar{n}_t$ and $\bar{m}_t$ are the average numbers of activated nodes and edges over time, resp. $\bar{C}_t = \bar{m}_t/\bar{n}_t$. The first 3 data are used for link prediction, and the others are for node classification.

Datasets	n	m	T	L	$\lfloor \bar{n}_t \rfloor$	$\bar{C}_t$
BitcoinAlpha	3,783	31,748	138	2	105	2.2
WikiElec	7,125	212,854	100	2	354	6.0
RedditBody	35,776	484,460	88	2	2,465	2.2
Brain	5,000	1,955,488	12	10	5,000	32.6
DBLP-3	4,257	23,540	10	3	782	3.0
DBLP-5	6,606	42,815	10	5	1,212	3.5
Reddit	8,291	264,050	10	4	2,071	12.8

Experiment

In this section, we evaluate TIARA to show its effectiveness for the augmentation problem for dynamic graphs.

Experimental Setting

Datasets. Table 1 summarizes 7 public datasets used in this work. BitcoinAlpha is a social network between bitcoin users (Kumar et al. 2016, 2018b). WikiElec is a voting network for Wikipedia adminship elections (Leskovec, Huttenlocher, and Kleinberg 2010). RedditBody is a hyperlink network of connections between two subreddits (Kumar et al. 2018a). For node classification, we use the following datasets evaluated in (Xu et al. 2019). Brain is a network of brain tissues where edges indicate their connectivities. DBLP-3 and DBLP-5 are co-authorship networks extracted from DBLP. Reddit is a post network where two posts were connected if they contain similar keywords.

Baseline augmentation methods. We compare TIARA to the following baselines. NONE indicates the result of a model without any augmentation. DROPEDGE is a drop-based method randomly removing edges at each epoch. GDC is a graph diffusion-based method where we use PPR

Table 2: Temporal link prediction accuracy (AUC) where NONE is a result without augmentation, and ▲ (or ▼) indicates improvement (or degradation) compared to None. TIARA shows consistent improvement across most models and datasets.

AUC	BitcoinAlpha			WikiElec			RedditBody		
	GCN	GCRN	EGCN	GCN	GCRN	EGCN	GCN	GCRN	EGCN
NONE	57.3±1.6	80.3±6.0	58.8±1.1	59.9±0.9	72.1±2.4	66.9±3.7	77.6±0.4	88.9±0.3	77.6±0.2
DROPEDGE	▼56.3±1.0	▼73.9±2.2	▼57.4±0.9	▼50.1±1.0	▼56.0±9.3	▼47.9±6.4	▼73.0±0.4	▼77.0±1.7	▼71.9±0.7
GDC	▲57.5±1.6	▼77.3±6.5	▼57.4±1.2	▲62.8±0.8	▼67.9±1.0	▼63.1±0.7	▼74.6±0.0	▼86.4±0.3	▼73.8±0.3
MERGE	▲66.8±2.6	▲93.1±0.4	▲61.0±9.2	▲60.6±1.7	▼68.4±3.2	▼60.7±1.3	▼69.7±0.7	▲89.8±0.5	▲80.3±0.5
TIARA	**▲76.0±1.3**	**▲94.6±0.8**	**▲77.2±1.4**	**▲69.0±1.2**	**▲73.4±2.2**	**▲69.1±0.3**	**▲80.8±0.6**	**▲90.2±0.4**	**▲82.0±0.1**

for this as our approach is based on random walks. MERGE is a simple baseline merging adjacency matrices from time 1 to t when training a model at time t. We apply DROPEDGE and GDC to each snapshot since they are designed for a static graph.

Baseline GNNs. We use GCN (Kipf and Welling 2017), GCRN (Seo et al. 2018) and EvolveGCN (Pareja et al. 2020), abbreviated to EGCN, for performing dynamic graph tasks. We naively apply a static GCN to each graph snapshot for verifying how temporal information is informative. We choose GCRN and EvolveGCN, lightweight and popular dynamic GNN models showing decent performance, to observe practical gains from augmentation. We adopt GCN layers for GCRN's graph convolution. We use the implementation of (Rozemberczki et al. 2021) for GCRN and EGCN. Note that any GNN models following Problem 1 can utilize TIARA because our approach is model-agnostic.

Training details. For each dataset, we tune the hyperparameters of all models on the original graph (marked as NONE) and augmented graphs separately through a combination of grid and random search on a validation set, and report test accuracy at the best validation epoch. For TIARA, we fix K to 100, search for ϵ in $[0.0001, 0.01]$, and tune α and β in $(0, 1)$ s.t. $0 < \alpha + \beta < 1$. We use the Adam optimizer with weight decay 10^{-4}, and the learning rate is tuned in $[0.01, 0.05]$ with decay factor 0.999. The dropout ratio is searched in $[0, 0.5]$. We repeat each experiment 5 times with different random seeds, and report the average and standard deviation of test values. We use PyTorch and DGL (Wang et al. 2019) to implement all methods. All experiments were done at workstations with Intel Xeon 4215R and RTX 3090. Details about the experimental setting are provided in Appendix D.

Temporal Link Prediction Task

This aims to predict whether an edge exists or not at time $t + 1$ using the information up to time t. As a standard setting (Pareja et al. 2020), we follow a chronological split with ratios of training (70%), validation (10%), and test (20%) sets. We sample the same amount of negative samples (edges) to positive samples (edges) for each time, and use AUC as a representative measure. We set the number of epochs to 200 with early stopping of patience 50.

As shown in Table 2, TIARA consistently improves the performance of dynamic GNN models such as GCRN and EGCN compared to NONE (i.e., without augmentation) while static augmentations of DROPEDGE and GDC do not. TIARA also outperforms the static methods on all models and datasets. This indicates it is not beneficial to only spatially augment the graphs for this task. TIARA even improves static GCN, which is competitive with EGCN, implying that effectively and temporally augmented data can even make static GNNs learn dynamic graphs well. In addition, MERGE also improves the accuracy of the tested models on many datasets. This confirms the need to utilize temporal information when it comes to dynamic graph augmentation in this task. However, MERGE performs worse than TIARA in most cases because TIARA can effectively augment both spatial and temporal localities at once while MERGE does not have a mechanism to enhance such localities.

Node Classification Task

This is to classify a label of each node where a graph and features change over time. Following (Xu et al. 2019), we split all nodes into training, validation, and test sets by the 7:1:2 ratio. We feed node embeddings $\mathbf{H}_T$ of each model forward to a softmax classifier, and use Macro F1-score because labels are imbalanced in each dataset. We set the number of epochs to $1,000$ with early stopping of patience 100.

Table 3 shows TIARA consistently improves the accuracies of GNNs on most datasets. Especially, TIARA significantly enhances the accuracies on the Brain dataset as another diffusion method GDC does, but TIARA shows better accuracy than GDC, implying it is effective to augment a temporal locality for the performance. For the other datasets, TIARA slightly improves each model, but it overall performs better than other augmentations. Note GCN and EGCN are worse than a random classifier of $1/L$ score (0.25 for $L=4$) in the Reddit where L is the number of labels, and all tested augmentations fail to beat the score, implying even these augmentations could not boost a poor model in this task.

Effect of Hyperparameters

We analyze the effects of temporal decay ratio γ and filtering threshold ϵ that mainly affect TIARA's results. We fix the number K of iterations to 100 for the power iteration, which leads to sufficiently accurate results for $\boldsymbol{\mathcal{L}}_t^{\text{rwr}}$.

Effect of the temporal decay ratio γ. As TIARA's hyperparameters, α and β should be analyzed, but our preliminary experiments showed that patterns vary by models and

Table 3: Node classification accuracy (Macro F1-score) where NONE is a result without augmentation, and ▲ (or ▼) indicates improvement (or degradation) compared to None. TIARA shows consistent improvement across most models and datasets.

Macro F1	**Brain**			**Reddit**			**DBLP-3**			**DBLP-5**		
	GCN	**GCRN**	**EGCN**	**GCN**	**GCRN**	**EGCN**	**GCN**	**GCRN**	**EGCN**	**GCN**	**GCRN**	**EGCN**
NONE	44.7±0.8	66.8±1.0	43.4±0.7	18.2±2.9	40.4±1.6	18.6±2.3	53.4±2.6	83.1±0.6	51.3±2.7	69.6±0.9	75.4±0.7	68.5±0.6
DROPEDGE	▼35.2±1.7	▲67.8±0.6	▼39.7±1.8	▲**19.4±0.8**	▼40.3±1.4	▼18.0±2.7	▲55.8±1.9	▲84.3±0.6	▲52.4±1.7	▲70.5±0.5	▲75.6±0.7	▼68.0±0.7
GDC	▲63.2±1.2	▲88.0±1.5	▲67.3±1.3	▼17.5±2.3	▲41.0±1.6	▼18.5±2.8	▲53.4±2.1	▲84.7±0.5	▲52.8±2.2	▲70.0±0.7	▲75.5±1.2	▲69.1±1.0
MERGE	▼34.4±3.4	▼63.2±1.6	▲53.0±0.9	▲19.3±3.0	▼39.6±0.8	▲20.4±3.0	▲54.9±3.1	▼83.0±1.4	▲53.3±1.2	▲70.8±0.4	▼74.5±0.8	▲69.7±1.6
TIARA	▲**68.7±1.2**	▲**91.3±1.0**	▲**72.0±0.6**	▲18.4±3.0	▲**41.5±1.5**	▲**21.9±1.6**	▲**57.5±2.2**	▲**84.9±1.6**	▲**56.4±1.8**	▲**71.1±0.6**	▲**77.9±0.4**	▲**70.1±1.0**

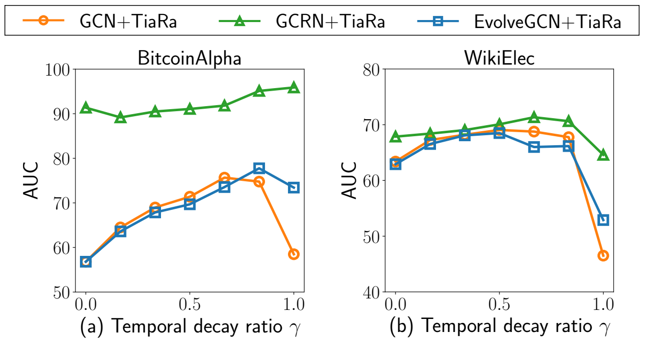


Figure 3: Effect of the temporal decay ratio γ.

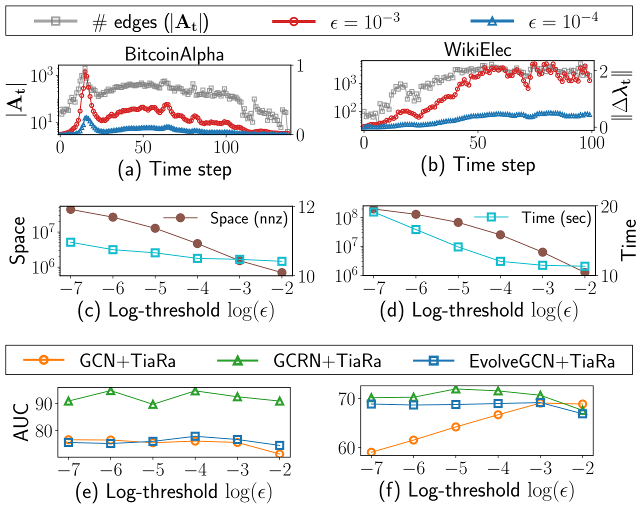


Figure 4: Effect of the filtering threshold ϵ.

datasets in the changes of α and β. Instead, we narrow our focus to γ in Equation (4) where $\gamma = \beta/(\alpha+\beta)$. For this experiment, we vary γ from 10^{-5} to $1-10^{-5}$ by tweaking α and β s.t. $\alpha+\beta$ is fixed to 0.6. Figure 3 shows that too small or large values of γ can degrade link prediction accuracy except GCRN with TIARA in BitcoinAlpha. This implies that it is important to properly mix spatial and temporal information about the performance, which is controlled by γ.

Effect of the filtering threshold ϵ. Figure 4 shows the effects of ϵ in terms of approximate error, time, space, and accuracy of link prediction in BitcoinAlpha and WikiElec. We fix α and β to 0.25, and vary ϵ from 10^{-7} to 10^{-2} for this experiment.

We measure the approximate error $\|\Delta\boldsymbol{\lambda}_t\| = \|\boldsymbol{\lambda}_t - \tilde{\boldsymbol{\lambda}}_t\|_2$ of eigenvalues where $\boldsymbol{\lambda}_t$ and $\tilde{\boldsymbol{\lambda}}_t$ are vectors of eigenvalues of $\boldsymbol{\mathcal{X}}_t$ (i.e., $\epsilon = 0$) and $\tilde{\boldsymbol{\mathcal{X}}}_t$, respectively, as similarly analyzed in (Klicpera, Weißenberger, and Günnemann 2019). The right y-axis of Figures 4(a) and (b) is the error, and the left y-axis is the number $|\mathbf{A}_t|$ of edges in $\mathbf{A}_t$. As time t increases, the errors (red and blue lines) remain small, and do not explode, implying errors incurred by repeated sparsifications are not excessively accumulated over time. Rather, the errors tend to be proportional to $|\mathbf{A}_t|$ at each time.

Figures 4 (c) and (d) show the space measured by $\sum_t |\tilde{\boldsymbol{\mathcal{X}}}_t|$ (left y-axis) and the augmentation time (right y-axis) of TIARA by ϵ between 10^{-7} and 10^{-2}. As the strength of sparsification increases (i.e., ϵ becomes larger), the produced non-zeros and the augmentation time decrease. On the other hand, most of the accuracies remain similar except $\epsilon = 10^{-2}$ as shown in Figures 4(e) and (f). Note it is not effective to truncate too many entries (e.g., $\epsilon = 10^{-2}$), or too dense $\tilde{\boldsymbol{\mathcal{X}}}_t$ can worse the performance as GCN in WikiElec. Thus, the sparsification with proper ϵ such as 10^{-3} or 10^{-4} provides a good trade-off between error, time, space, and accuracy.

Conclusion

In this work, we propose TIARA, a novel and model-agnostic diffusion method for augmenting a dynamic graph with the purpose of improvements in dynamic GNN models. We first extend Random Walk with Restart (RWR) to Time-aware RWR so that it produces spatially and temporally localized scores. We then formulate time-aware random walk diffusion matrices, and analyze how our diffusion approach augments both spatial and temporal localities in the dynamic graph. As graph diffusions lead to dense matrices, we further employ approximate techniques such as power iteration and sparsification, and analyze how they are effective for achieving a good trade-off between error, time, space, and predictive accuracy. Our experiments on various real-world dynamic graphs show that TIARA aids GNN models in providing better performance of temporal link prediction and node classification tasks.

References

Andersen, R.; Chung, F.; and Lang, K. 2006. Local graph partitioning using pagerank vectors. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. IEEE.

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Advances in Neural Information Processing Systems*.

Feng, W.; Zhang, J.; Dong, Y.; Han, Y.; Luan, H.; Xu, Q.; Yang, Q.; Kharlamov, E.; and Tang, J. 2020. Graph random neural networks for semi-supervised learning on graphs. *Advances in neural information processing systems*.

Guo, S.; Lin, Y.; Feng, N.; Song, C.; and Wan, H. 2019. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *Proceedings of the AAAI conference on artificial intelligence*.

Han, Y.; Huang, G.; Song, S.; Yang, L.; Wang, H.; and Wang, Y. 2021. Dynamic neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Hou, G.; Chen, X.; Wang, S.; and Wei, Z. 2021. Massively parallel algorithms for personalized pagerank. *Proceedings of the VLDB Endowment*, 14(9): 1668–1680.

Huang, Z.; Sun, Y.; and Wang, W. 2021. Coupled Graph ODE for Learning Interacting System Dynamics. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, 705–715. ACM.

Jung, J.; Jin, W.; Sael, L.; and Kang, U. 2016. Personalized Ranking in Signed Networks Using Signed Random Walk with Restart. In *IEEE 16th International Conference on Data Mining, ICDM*, 973–978. IEEE Computer Society.

Jung, J.; Jung, J.; and Kang, U. 2021. Learning to Walk across Time for Interpretable Temporal Knowledge Graph Completion. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 786–795. ACM.

Jung, J.; Park, N.; Sael, L.; and Kang, U. 2017. BePI: Fast and Memory-Efficient Method for Billion-Scale Random Walk with Restart. In *Proceedings of the 2017 ACM SIGMOD International Conference on Management of Data*, 789–804. ACM.

Jung, J.; Yoo, J.; and Kang, U. 2022. Signed random walk diffusion for effective representation learning in signed graphs. *Plos one*, 17(3): e0265001.

Kazemi, S. M.; Goel, R.; Jain, K.; Kobyzev, I.; Sethi, A.; Forsyth, P.; and Poupart, P. 2020. Representation learning for dynamic graphs: A survey. *J. Mach. Learn. Res.*, 21(70).

Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR)*.

Klicpera, J.; Weißenberger, S.; and Günnemann, S. 2019. Diffusion improves graph learning. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*.

Kumar, S.; Hamilton, W. L.; Leskovec, J.; and Jurafsky, D. 2018a. Community interaction and conflict on the web. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*. International World Wide Web Conferences Steering Committee.

Kumar, S.; Hooi, B.; Makhija, D.; Kumar, M.; Faloutsos, C.; and Subrahmanian, V. 2018b. Rev2: Fraudulent user prediction in rating platforms. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*.

Kumar, S.; Spezzano, F.; Subrahmanian, V.; and Faloutsos, C. 2016. Edge weight prediction in weighted signed networks. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, 221–230. IEEE.

Lee, D.; Shin, K.; and Faloutsos, C. 2020. Temporal locality-aware sampling for accurate triangle counting in real graph streams. *VLDB J.*, 29(6): 1501–1525.

Leskovec, J.; Huttenlocher, D.; and Kleinberg, J. 2010. Predicting positive and negative links in online social networks. In *Proceedings of the 19th international conference on World wide web*.

Nassar, H.; Kloster, K.; and Gleich, D. F. 2015. Strong localization in personalized PageRank vectors. In *International Workshop on Algorithms and Models for the Web-Graph*. Springer.

Panagopoulos, G.; Nikolentzos, G.; and Vazirgiannis, M. 2021. Transfer graph neural networks for pandemic forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Pareja, A.; Domeniconi, G.; Chen, J.; Ma, T.; Suzumura, T.; Kanezashi, H.; Kaler, T.; Schardl, T. B.; and Leiserson, C. E. 2020. EvolveGCN: Evolving Graph Convolutional Networks for Dynamic Graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Raghavendra, M.; Sharma, K.; Kumar, S.; et al. 2022. Signed Link Representation in Continuous-Time Dynamic Signed Networks. *arXiv preprint arXiv:2207.03408*.

Rong, Y.; Huang, W.; Xu, T.; and Huang, J. 2020. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In *International Conference on Learning Representations*.

Rossi, E.; Chamberlain, B.; Frasca, F.; Eynard, D.; Monti, F.; and Bronstein, M. 2020. Temporal Graph Networks for Deep Learning on Dynamic Graphs. In *ICML 2020 Workshop on Graph Representation Learning*.

Rozemberczki, B.; Scherer, P.; He, Y.; Panagopoulos, G.; Riedel, A.; Astefanoaei, M.; Kiss, O.; Beres, F.; ; Lopez, G.; Collignon, N.; and Sarkar, R. 2021. PyTorch Geometric Temporal: Spatiotemporal Signal Processing with Neural Machine Learning Models. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management*, 4564–4573.

Sankar, A.; Wu, Y.; Gou, L.; Zhang, W.; and Yang, H. 2020. DySAT: Deep Neural Representation Learning on Dynamic Graphs via Self-Attention Networks. In *Proceedings of the 13th International Conference on Web Search and Data Mining*.

Seo, Y.; Defferrard, M.; Vandergheynst, P.; and Bresson, X. 2018. Structured sequence modeling with graph convolutional recurrent networks. In *International conference on neural information processing*. Springer.

Shin, K. 2017. Wrs: Waiting room sampling for accurate triangle counting in real graph streams. In *2017 IEEE International Conference on Data Mining (ICDM)*. IEEE.

Shin, K.; Jung, J.; Sael, L.; and Kang, U. 2015. BEAR: Block Elimination Approach for Random Walk with Restart on Large Graphs. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 1571–1585. ACM.

Skarding, J.; Gabrys, B.; and Musial, K. 2021. Foundations and modeling of dynamic networks using dynamic graph neural networks: A survey. *IEEE Access*.

Srivastava, N.; Jin, H.; Liu, J.; Albonesi, D.; and Zhang, Z. 2020. Matraptor: A sparse-sparse matrix multiplication accelerator based on row-wise product. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture*.

Tong, H.; Faloutsos, C.; and Pan, J.-Y. 2006. Fast random walk with restart and its applications. In *Sixth international conference on data mining (ICDM'06)*. IEEE.

Wang, M.; Zheng, D.; Ye, Z.; Gan, Q.; Li, M.; Song, X.; Zhou, J.; Ma, C.; Yu, L.; Gai, Y.; Xiao, T.; He, T.; Karypis, G.; Li, J.; and Zhang, Z. 2019. Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks. *arXiv preprint arXiv:1909.01315*.

Wang, S.; Yang, R.; Xiao, X.; Wei, Z.; and Yang, Y. 2017. FORA: simple and effective approximate single-source personalized pagerank. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 505–514.

Wang, Y.; Cai, Y.; Liang, Y.; Ding, H.; Wang, C.; Bhatia, S.; and Hooi, B. 2021. Adaptive data augmentation on temporal graphs. *Advances in Neural Information Processing Systems*.

Wu, Z.; Pan, S.; Long, G.; Jiang, J.; Chang, X.; and Zhang, C. 2020. Connecting the dots: Multivariate time series forecasting with graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*.

Xu, D.; Cheng, W.; Luo, D.; Liu, X.; and Zhang, X. 2019. Spatio-Temporal Attentive RNN for Node Classification in Temporal Attributed Graphs. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI*.

Xu, D.; Ruan, C.; Körpeoglu, E.; Kumar, S.; and Achan, K. 2020. Inductive representation learning on temporal graphs. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*.

Yang, M.; Zhou, M.; Kalander, M.; Huang, Z.; and King, I. 2021. Discrete-time temporal network embedding via implicit hierarchical learning in hyperbolic space. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*.

Yoo, J.; Jeon, H.; Jung, J.; and Kang, U. 2022. Accurate Node Feature Estimation with Structured Variational Graph Autoencoder. In *KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2336–2346. ACM.

Yoo, J.; Shim, S.; and Kang, U. 2022. Model-Agnostic Augmentation for Accurate Graph Classification. In *WWW '22: The ACM Web Conference 2022*, 1281–1291. ACM.

Yoon, M.; Jung, J.; and Kang, U. 2018. Tpa: Fast, scalable, and accurate method for approximate random walk with restart on billion scale graphs. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE.

Yu, B.; Yin, H.; and Zhu, Z. 2018. Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting. In *IJCAI*.

Zhao, T.; Liu, G.; Günnemann, S.; and Jiang, M. 2022. Graph Data Augmentation for Graph Machine Learning: A Survey. *arXiv preprint arXiv:2202.08871*.

Table 4: Table of symbols.

Symbol	Definition
$\mathcal{G} = \{\mathcal{G}_1, \cdots, \mathcal{G}_T\}$	discrete-time dynamic graph
$\mathcal{G}_t = (\mathcal{V}, \mathcal{E}_t, \mathbf{F}_t)$	graph snapshot at time t of $\mathcal{G}$
$\mathcal{E}_t$	set of edges at time t
$\mathcal{V}$	set of nodes where $n = \lvert\mathcal{V}\rvert$
$\mathcal{V}_t \subset \mathcal{V}$	set of activated nodes where $n_t = \lvert\mathcal{V}_t\rvert$
$\mathbf{F}_t \in \mathbb{R}^{n \times d}$	initial node feature matrix at time t where d is the number of features
$\mathbf{A}_t \in \mathbb{R}^{n \times n}$	adjacency matrix of $\mathcal{G}_t$
$\tilde{\boldsymbol{\mathcal{A}}}_t \in \mathbb{R}^{n \times n}$	normalized adjacency matrix of $\mathcal{G}_t$
$\mathcal{F}_\Theta(\cdot)$	GNN for discrete-time dynamic graphs
$\boldsymbol{\mathcal{I}}_n \in \mathbb{R}^{n \times n}$	identity matrix
α	restart probability
β	time travel probability
K	number of power iterations
ϵ	filtering threshold
γ	temporal decay ratio $\gamma = \beta/(\alpha + \beta)$
$\mathbf{L}_t \in \mathbb{R}^{n \times n}$	$Łt = \boldsymbol{\mathcal{I}}_n - (1 - \alpha - \beta)\tilde{\boldsymbol{\mathcal{A}}}_t^\top$
$\boldsymbol{\mathcal{L}}_t^{\text{rwr}} \in \mathbb{R}^{n \times n}$	RWR diffusion kernel on the graph $\mathcal{G}_t$ where $\boldsymbol{\mathcal{L}}_t^{\text{rwr}} = (\alpha + \beta)Łt^{-1}$
$\boldsymbol{\mathcal{S}}_t \in \mathbb{R}^{n \times n}$	spatial diffusion matrix
$\boldsymbol{\mathcal{T}}_t \in \mathbb{R}^{n \times n}$	temporal diffusion matrix
$\boldsymbol{\mathcal{X}}_t \in \mathbb{R}^{n \times n}$	time-aware random walk diffusion matrix

A Proofs

Lemma 1. *For every time step t, $\boldsymbol{\mathcal{X}}_t$ is column stochastic.*

Proof. As a base case, $\boldsymbol{\mathcal{X}}_0 = \boldsymbol{\mathcal{I}}_n$ for $t = 0$, which is trivially column stochastic. Assume $\boldsymbol{\mathcal{X}}_{t-1}$ is column stochastic, i.e., $\mathbf{1}^\top\boldsymbol{\mathcal{X}}_{t-1} = \mathbf{1}^\top$ where $\mathbf{1} \in \mathbb{R}^n$ is a column vector of ones. Then, $\mathbf{1}^\top\boldsymbol{\mathcal{X}}_t$ is written as follows:

$$\begin{aligned}\mathbf{1}^\top\boldsymbol{\mathcal{X}}_t &= (1-\gamma)\mathbf{1}^\top\boldsymbol{\mathcal{L}}_t^{\text{rwr}}\boldsymbol{\mathcal{I}}_n + \gamma\mathbf{1}^\top\boldsymbol{\mathcal{L}}_t^{\text{rwr}}\boldsymbol{\mathcal{X}}_{t-1} \\ &= (1-\gamma)\mathbf{1}^\top\boldsymbol{\mathcal{I}}_n + \gamma\mathbf{1}^\top\boldsymbol{\mathcal{X}}_{t-1} \\ &= (1-\gamma)\mathbf{1}^\top + \gamma\mathbf{1}^\top = \mathbf{1}^\top\end{aligned}$$

where $\mathbf{1}^\top\boldsymbol{\mathcal{L}}_t^{\text{rwr}} = \mathbf{1}^\top$. Therefore, the claim holds for every $t \geq 0$ by mathematical induction. □

Proof of Theorem 1. We prove Theorem 1 as follows:

Proof. We begin the derivation from the following equation:

$$\boldsymbol{\mathcal{X}}_{t+1} = (1-\gamma)\left(\boldsymbol{\mathcal{L}}_{t+1}^{\text{rwr}}\boldsymbol{\mathcal{I}}_n\right) + \gamma\left(\boldsymbol{\mathcal{L}}_{t+1}^{\text{rwr}}\boldsymbol{\mathcal{X}}_t\right) \tag{4}$$

As a base case, $\boldsymbol{\mathcal{X}}_1$ is ($t = 0$ in the above) as follows:

$$\boldsymbol{\mathcal{X}}_1 = (1-\gamma)\left(\boldsymbol{\mathcal{L}}_1^{\text{rwr}}\boldsymbol{\mathcal{I}}_n\right) + \gamma\left(\boldsymbol{\mathcal{L}}_1^{\text{rwr}}\boldsymbol{\mathcal{X}}_0\right) = \boldsymbol{\mathcal{L}}_1^{\text{rwr}}$$

where $\boldsymbol{\mathcal{X}}_0 = \boldsymbol{\mathcal{I}}_n$. This trivially holds Equation (5). Let's assume that Equation (5) holds at t. Then, by substituting $\boldsymbol{\mathcal{X}}_t$ of Equation (5) into Equation (4), $\boldsymbol{\mathcal{X}}_{t+1}$ is:

$$\begin{aligned}\boldsymbol{\mathcal{X}}_{t+1} &= (1-\gamma)\left(\boldsymbol{\mathcal{L}}_{t+1}^{\text{rwr}} + \sum_{i=0}^{t-2}\gamma^{i+1}\boldsymbol{\mathcal{L}}_{t+1\leftsquigarrow t-i}^{\text{rwr}}\right) + \gamma^t\boldsymbol{\mathcal{L}}_{t+1\leftsquigarrow 1}^{\text{rwr}} \\ &= (1-\gamma)\left(\sum_{i=0}^{t-1}\gamma^i\boldsymbol{\mathcal{L}}_{t+1\leftsquigarrow t+1-i}^{\text{rwr}}\right) + \gamma^t\boldsymbol{\mathcal{L}}_{t+1\leftsquigarrow 1}^{\text{rwr}}\end{aligned}$$

Suppose $k = t + 1$; then, $\boldsymbol{\mathcal{X}}_k$ is represented as follows:

$$\boldsymbol{\mathcal{X}}_k = (1-\gamma)\left(\sum_{i=0}^{k-2}\gamma^i\boldsymbol{\mathcal{L}}_{k\leftsquigarrow k-i}^{\text{rwr}}\right) + \gamma^{k-1}\boldsymbol{\mathcal{L}}_{k\leftsquigarrow 1}^{\text{rwr}}$$

Note that the equation of $\boldsymbol{\mathcal{X}}_k$ has the same form of that of $\boldsymbol{\mathcal{X}}_t$ in Equation (4). This indicates Equation (4) also holds at $k = t + 1$. Thus, the claim holds for every $t \geq 1$ by mathematical induction. □

Lemma 2. *Suppose $c = 1 - \alpha - \beta$ and $0 < c < 1$. The approximate error of the following power iteration is bounded by c^k, and converges to 0 as $k \to \infty$:*

$$\mathbf{M}_t^{(k)} = \boldsymbol{\mathcal{I}}_n + c\tilde{\boldsymbol{\mathcal{A}}}_t^\top\mathbf{M}_t^{(k-1)}$$

where $\tilde{\boldsymbol{\mathcal{A}}}_t$ is column stochastic, and $\mathbf{M}_t^{(0)} = \boldsymbol{\mathcal{I}}_n$.

Proof. Let $\mathbf{M}_t^*$ be the stationary matrix of the equation. Then, the iteration is represented as $\mathbf{M}_t^* = \boldsymbol{\mathcal{I}}_n + c\tilde{\boldsymbol{\mathcal{A}}}_t^\top\mathbf{M}_t^*$, implying $\mathbf{M}_t^* = \mathbf{L}_t^{-1}$. Then, the error $\|\mathbf{M}_t^* - \mathbf{M}_t^{(k)}\|_1$ is represented as follows:

$$\begin{aligned}\|\mathbf{M}_t^* - \mathbf{M}_t^{(k)}\|_1 &= \|c\tilde{\boldsymbol{\mathcal{A}}}_t^\top\mathbf{M}_t^* - c\tilde{\boldsymbol{\mathcal{A}}}_t^\top\mathbf{M}_t^{(k-1)}\|_1 \\ &\leq c\|\tilde{\boldsymbol{\mathcal{A}}}_t^\top\|_1\|\mathbf{M}_t^* - \mathbf{M}_t^{(k-1)}\|_1 \\ &\cdots \\ &\leq c^k\|\mathbf{M}_t^* - \mathbf{M}_t^{(0)}\|_1 = c^k\|\mathbf{M}_t^* - \boldsymbol{\mathcal{I}}_n\|_1 \\ &\leq c^k\left(\frac{c}{1-c}\right)\end{aligned}$$

where $\|\cdot\|_1$ is L1 norm of a matrix. Note $\|(1-c)\mathbf{L}_t^{-1}\|_1 = 1$ since $(1-c)\mathbf{L}_t^{-1}$ is stochastic, and $\mathbf{M}_t^* = \mathbf{L}_t^{-1}$.

Hence, $\|\mathbf{M}_t^*\|_1 = (1-c)^{-1}$. Then, each column sum of $\mathbf{M}_t^* - \boldsymbol{\mathcal{I}}_n$ is $(1-c)^{-1} - 1$; thus, $\|\mathbf{M}_t^* - \boldsymbol{\mathcal{I}}_n\|_1 = c/(1-c)$. Since $c/(1-c)$ is a constant, the error converges to 0 as $k \to \infty$. □

Proof of Theorem 2. We prove Theorem 2 as follows:

Proof. According to Lemma 3, nnz$(\tilde{\boldsymbol{\mathcal{X}}}_t) = O(n/\epsilon)$. Thus, the space complexity of $\tilde{\boldsymbol{\mathcal{X}}}_t$ is $O(n/\epsilon)$ if we use a sparse matrix format such as CSR or CSC. The time complexity is proved in Lemma 4. □

Lemma 3. *The sparsification truncates small entries of $\boldsymbol{\mathcal{X}}_t$ if their weights are $< \epsilon$. Then, $\tilde{\boldsymbol{\mathcal{X}}}_t$ has n/ϵ non-zeros at most, i.e., nnz$(\tilde{\boldsymbol{\mathcal{X}}}_t) = O(n/\epsilon)$ after the sparsification.*

Proof. Let $\tilde{\mathbf{x}}_{t,s}$ denote the s-th column of $\tilde{\boldsymbol{\mathcal{X}}}_t$. Then, the claim is represented as nnz$(\tilde{\mathbf{x}}_{t,s}) \leq \frac{1}{\epsilon}$, and proved by contradiction. Let's say the claim is false, i.e., nnz$(\tilde{\mathbf{x}}_{t,s}) > \frac{1}{\epsilon}$. Assume $\tilde{\mathbf{x}}_{t,s}$ has $\frac{1}{\epsilon} + 1$ non-zeros after the sparsification. This implies their weights were $\geq \epsilon$, and the sum of their weights was $\geq (\frac{1}{\epsilon} + 1)\epsilon = 1 + \epsilon$ before the sparsification. For $0 < \epsilon < 1$, this contradicts to the fact that $\boldsymbol{\mathcal{X}}_t$ is column stochastic, i.e., each column sum of $\boldsymbol{\mathcal{X}}_t$ is 1. Therefore, nnz$(\tilde{\mathbf{x}}_{t,s}) \leq \frac{1}{\epsilon}$ after the sparsification, and it holds for every column. Thus, nnz$(\tilde{\boldsymbol{\mathcal{X}}}_t) \leq n/\epsilon = O(n/\epsilon)$. □

Table 5: Time complexity of Algorithm 1

Line	Task	Time Complexity
1	normalize $\mathbf{A}_t$	$O(m_t + n)$
2	power-iterate $\boldsymbol{\mathcal{L}}_t^{\text{rwr}}$	$O(n_t^2 K + nK)$
4	$\boldsymbol{\mathcal{T}}_t \leftarrow \boldsymbol{\mathcal{S}}_t \tilde{\boldsymbol{\mathcal{X}}}_t$	$O(nn_t/\epsilon)$
5	$\boldsymbol{\mathcal{X}}_t \leftarrow (1-\gamma)\boldsymbol{\mathcal{S}}_t + \gamma \boldsymbol{\mathcal{T}}_t$	$O(nn_t + n/\epsilon)$
6	filter $\boldsymbol{\mathcal{X}}_t$ into $\tilde{\boldsymbol{\mathcal{X}}}_t$	$O(nn_t + n/\epsilon)$
7	normalize $\tilde{\boldsymbol{\mathcal{X}}}_t$	$O(n/\epsilon)$
Total		$O(n_t n/\epsilon + n_t^2 K)$

Lemma 4. *For each time step t, Algorithm 1 of* TiaRa *takes $O(n_t n/\epsilon + n_t^2 K)$ time in expectation.*

Proof. Table 5 summarizes the time complexity of each step of TiaRa. Line 1 normalizes the self-looped adjacency matrix $\mathbf{A}_t$. Let m_t be the number of edges in $\mathbf{A}_t$ without self-loops, and n be the number of all nodes. Then, it takes $O(m_t + n)$ to obtain $\mathbf{D}_t^{-1}$ and multiply it and $\mathbf{A}_t$.

Line 2 performs the power-iteration to compute $\boldsymbol{\mathcal{L}}_t^{\text{rwr}}$. Let $\mathcal{V}_t$ denote the set of activated nodes, and $n_t = |\mathcal{V}_t|$ at time t. Then, $\tilde{\boldsymbol{\mathcal{A}}}_t^\top$ is reordered by activated and non-activated nodes as follows:

$$\tilde{\boldsymbol{\mathcal{A}}}_t^\top = \begin{bmatrix} \tilde{\boldsymbol{\mathcal{A}}}_{\mathcal{V}_t}^\top & \mathbf{O} \\ \mathbf{O}^\top & \boldsymbol{\mathcal{I}}_{n-n_t} \end{bmatrix}$$

where $\mathbf{O} \in \mathbb{R}^{n_t \times n - n_t}$ is a zero matrix, and $\tilde{\boldsymbol{\mathcal{A}}}_{\mathcal{V}_t} \in \mathbb{R}^{n_t \times n_t}$ is the normalized adjacency matrix for the activated nodes in $\mathcal{V}_t$. Then, the k-th iteration is represented as follows:

$$\mathbf{M}_t^{(k)} = \boldsymbol{\mathcal{I}}_n + c\tilde{\boldsymbol{\mathcal{A}}}_t^\top \mathbf{M}_t^{(k-1)}$$

$$\mathbf{M}_t^{(k)} = \begin{bmatrix} \boldsymbol{\mathcal{I}}_{n_t} & \mathbf{O} \\ \mathbf{O}^\top & \boldsymbol{\mathcal{I}}_{n-n_t} \end{bmatrix} + c \begin{bmatrix} \tilde{\boldsymbol{\mathcal{A}}}_{\mathcal{V}_t}^\top & \mathbf{O} \\ \mathbf{O}^\top & \boldsymbol{\mathcal{I}}_{n-n_t} \end{bmatrix} \begin{bmatrix} \mathbf{M}_{t,11}^{(k-1)} & \mathbf{M}_{t,12}^{(k-1)} \\ \mathbf{M}_{t,21}^{(k-1)} & \mathbf{M}_{t,22}^{(k-1)} \end{bmatrix}$$

$$\mathbf{M}_t^{(k)} = \begin{bmatrix} \boldsymbol{\mathcal{I}}_{n_t} + c\tilde{\boldsymbol{\mathcal{A}}}_{\mathcal{V}_t}^\top \mathbf{M}_{t,11}^{(k-1)} & c\tilde{\boldsymbol{\mathcal{A}}}_{\mathcal{V}_t}^\top \mathbf{M}_{t,12}^{(k-1)} \\ c\mathbf{M}_{t,21}^{(k-1)} & \boldsymbol{\mathcal{I}}_{n-n_t} + c\mathbf{M}_{t,22}^{(k-1)} \end{bmatrix}$$

Note that $\mathbf{M}_t^{(0)} = \boldsymbol{\mathcal{I}}_n$ at the beginning. Thus, $\mathbf{M}_{t,12}^{(i)}$ and $\mathbf{M}_{t,21}^{(i)}$ remain as zeros for every iteration i because they are off the diagonal. Therefore, $\mathbf{M}_t^{(k)}$ is written as follows:

$$\mathbf{M}_t^{(k)} = \begin{bmatrix} \boldsymbol{\mathcal{I}}_{n_t} + c\tilde{\boldsymbol{\mathcal{A}}}_{\mathcal{V}_t}^\top \mathbf{M}_{t,11}^{(k-1)} & \mathbf{O} \\ \mathbf{O}^\top & \boldsymbol{\mathcal{I}}_{n-n_t} + c\mathbf{M}_{t,22}^{(k-1)} \end{bmatrix}$$

where $\mathbf{M}_{t,22}^{(i)}$ remains diagonal as $\mathbf{M}_{t,22}^{(0)} = \boldsymbol{\mathcal{I}}_{n-n_t}$. In the equation, $\tilde{\boldsymbol{\mathcal{A}}}_{\mathcal{V}_t}^\top \mathbf{M}_{t,11}^{(k-1)}$ takes $O(|\tilde{\boldsymbol{\mathcal{A}}}_{\mathcal{V}_t}^\top| n_t)$ time where $\tilde{\boldsymbol{\mathcal{A}}}_{\mathcal{V}_t}$ is sparse (Lemma 5), and $|\tilde{\boldsymbol{\mathcal{A}}}_{\mathcal{V}_t}^\top| = m_t + n_t$ where m_t and n_t are the numbers of edges and self-loops, respectively. Also, in real-world dynamic graphs, $m_t = Cn_t$ and C is a constant. The average of m_t/n_t over time was between 2.2 and 32.6 in the datasets used in this work (see Table 1 of the submitted paper). Thus, $\tilde{\boldsymbol{\mathcal{A}}}_{\mathcal{V}_t}^\top \mathbf{M}_{t,11}^{(k-1)}$ takes $O(n_t^2)$ time on average. Also, it takes $O(n - n_t)$ time to compute $\boldsymbol{\mathcal{I}}_{n-n_t} + c\mathbf{M}_{t,22}^{(k-1)}$; each iteration takes $O(n_t^2 + n)$; thus, $O(n_t^2 K + nK)$ time is required for K iterations.

Line 4 computes $\boldsymbol{\mathcal{S}}_t \tilde{\boldsymbol{\mathcal{X}}}_t$ where $\boldsymbol{\mathcal{S}}_t = \boldsymbol{\mathcal{L}}_t^{\text{rwr}}$. Note that the structure of $\boldsymbol{\mathcal{L}}_t^{\text{rwr}}$ is the same as that of $\mathbf{M}_t^{(K)}$, which is a block diagonal matrix as shown in the above. Thus, $\boldsymbol{\mathcal{S}}_t \tilde{\boldsymbol{\mathcal{X}}}_t$ is represented as follows:

$$\boldsymbol{\mathcal{T}}_t = \boldsymbol{\mathcal{S}}_t \tilde{\boldsymbol{\mathcal{X}}}_t = \begin{bmatrix} \boldsymbol{\mathcal{S}}_{t,11} & \mathbf{O} \\ \mathbf{O}^\top & \boldsymbol{\mathcal{S}}_{t,22} \end{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{\mathcal{X}}}_{t,11} & \tilde{\boldsymbol{\mathcal{X}}}_{t,12} \\ \tilde{\boldsymbol{\mathcal{X}}}_{t,21} & \tilde{\boldsymbol{\mathcal{X}}}_{t,22} \end{bmatrix}$$

where $\boldsymbol{\mathcal{S}}_{t,22} = (1-c)\mathbf{M}_t^{(K)}$ converges to $\boldsymbol{\mathcal{I}}_{n-n_t}$ as K increases, and it is normalized (line 15). Then, the above is written as:

$$\boldsymbol{\mathcal{T}}_t = \boldsymbol{\mathcal{S}}_t \tilde{\boldsymbol{\mathcal{X}}}_t = \begin{bmatrix} \boldsymbol{\mathcal{S}}_{t,11} & \mathbf{O} \\ \mathbf{O}^\top & \boldsymbol{\mathcal{I}}_{n-n_t} \end{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{\mathcal{X}}}_{t,11} & \tilde{\boldsymbol{\mathcal{X}}}_{t,12} \\ \tilde{\boldsymbol{\mathcal{X}}}_{t,21} & \tilde{\boldsymbol{\mathcal{X}}}_{t,22} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\mathcal{S}}_{t,11}\tilde{\boldsymbol{\mathcal{X}}}_{t,11} & \boldsymbol{\mathcal{S}}_{t,11}\tilde{\boldsymbol{\mathcal{X}}}_{t,12} \\ \tilde{\boldsymbol{\mathcal{X}}}_{t,21} & \tilde{\boldsymbol{\mathcal{X}}}_{t,22} \end{bmatrix}$$

Note $\tilde{\boldsymbol{\mathcal{X}}}_t$ is sparse, and has $O(n/\epsilon)$ non-zeros (Lemma 3); thus, its sub-matrices have less non-zeros than that. To exploit the sparsity, suppose we compute $(\tilde{\boldsymbol{\mathcal{X}}}_{t,11}^\top \boldsymbol{\mathcal{S}}_{t,11}^\top)^\top$ and $(\tilde{\boldsymbol{\mathcal{X}}}_{t,12}^\top \boldsymbol{\mathcal{S}}_{t,11}^\top)^\top$. Then, they take $O(n_t n/\epsilon)$ time for sparse matrix multiplications (Lemma 5).

Lines 5 and 6 compute $\boldsymbol{\mathcal{X}}_t \leftarrow (1-\gamma)\boldsymbol{\mathcal{S}}_t + \gamma\boldsymbol{\mathcal{T}}_t$, and filter $\boldsymbol{\mathcal{X}}_t$. The worst number of non-zeros of $\boldsymbol{\mathcal{S}}_t$ is $O(n_t^2 + n)$ for $\boldsymbol{\mathcal{S}}_{t,11}$ and $\boldsymbol{\mathcal{S}}_{t,22}$. The worst number of non-zeros of $\boldsymbol{\mathcal{T}}_t$ is decided when $\boldsymbol{\mathcal{T}}_{t,11}$ and $\boldsymbol{\mathcal{T}}_{t,12}$ are fully dense; then, it is $O(nn_t + n/\epsilon)$. Thus, $O(nn_t + n/\epsilon)$ time is required to add them and filter $\boldsymbol{\mathcal{X}}_t$ because it is expected to be proportional to $\text{nnz}(\boldsymbol{\mathcal{X}}_t)$. After filtering, $\tilde{\boldsymbol{\mathcal{X}}}_t$ has $O(n/\epsilon)$ non-zeros by Lemma 3; it takes $O(n/\epsilon)$ time to normalize $\tilde{\boldsymbol{\mathcal{X}}}_t$. □

Lemma 5. *Let* $\mathbf{A}$ *and* $\mathbf{B}$ *be $p \times q$ and $q \times r$ matrices, respectively, and* $\mathbf{A}$ *has* $|\mathbf{A}|$ *non-zero entries. Then, the sparse matrix multiplication* $\mathbf{C} = \mathbf{AB}$ *takes* $O(|\mathbf{A}|r)$ *time.*

Proof. The i-th column of $\mathbf{C}$ is the result of a sparse matrix vector multiplication with $\mathbf{A}$ and the i-th column of $\mathbf{B}$, taking $O(|\mathbf{A}|)$ for a sparse matrix $\mathbf{A}$ in the CSR format. Thus, for r columns, it takes $O(|\mathbf{A}|r)$ time. □

B Discussions

We discuss several aspects about dynamic graph augmentation and our TiaRa in this section.

Insertion or deletion of edges. TiaRa naturally supports the operations of edge insertion and deletion on discrete-time dynamic graphs (DTDG). According to (Skarding, Gabrys, and Musial 2021), a DTDG is an ordered sequence of graph snapshots, and it can represent *temporal graphs* or *evolving graphs*. A temporal graph is a network of edges where they exist for a certain time range. In DTDGs, a set of edges in graph snapshot $\mathcal{G}_t$ only exists at time t. Thus, it is more natural to think of a link as an event with a duration (Skarding, Gabrys, and Musial 2021), rather than insertion or deletion. By default, TiaRa can augment a sequence of graph snapshots which represents such a temporal graph.

On the other hand, there are edge insertion and deletion in evolving graphs. An evolving graph keeps the overall structure by continually adding or deleting edges. Each snapshot also represents the adjacency matrix $\mathbf{A}_t$ at each time t, but in evolving graphs, $\mathbf{A}_t$ and $\mathbf{A}_{t+1}$ are mostly similar while they are different in temporal graphs. As the evolving graph supports the insertion and deletion on edges, TiaRa also naturally supports them. If an edge (u, v) is inserted or deleted at time t, we simply mark $\mathbf{A}_{t;uv}$ as 1 (insertion) or 0 (deletion), and then perform TiaRa on the graph snapshot.

Insertion or deletion of nodes. TiaRa can support node insertion and deletion after a few modifications. For the node insertion, suppose a node u is inserted at time t, i.e., the node did not exist before time t. Then, we assign the node u to a position corresponding to the last index of $\tilde{\boldsymbol{\mathcal{X}}}_{t-1}$ as follows:

$$\underbrace{\tilde{\boldsymbol{\mathcal{X}}}_{t-1}}_{\text{Before}} \xrightarrow{u \text{ is inserted}} \underbrace{\tilde{\boldsymbol{\mathcal{X}}}_{t-1} \leftarrow \begin{bmatrix} \tilde{\boldsymbol{\mathcal{X}}}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{bmatrix}}_{\text{After}}$$

where $\mathbf{0}$ is an n-dimensional column vector of size n. After then, TiaRa performs its diffusion with new $\mathbf{A}_t$ and $\tilde{\boldsymbol{\mathcal{X}}}_{t-1}$ in which the node u is inserted.

For the node deletion, suppose a node u is deleted at time t, i.e., the node will not exist from time t. For this, we delete the u-th row and column of $\tilde{\boldsymbol{\mathcal{X}}}_{t-1}$, and normalize the modified $\tilde{\boldsymbol{\mathcal{X}}}_{t-1}$ column-wise to make it column stochastic. After reordering node indices to match remaining nodes, TiaRa performs its diffusion with new $\mathbf{A}_t$ and $\tilde{\boldsymbol{\mathcal{X}}}_{t-1}$ in which the node u is deleted.

Discussion on other types of dynamic GNNs. There is another type of dynamic graphs, called traffic networks (e.g., roads), where the structure of a graph is static while node features vary over time. For learning traffic networks, many models have been proposed (Yu, Yin, and Zhu 2018; Guo et al. 2019; Wu et al. 2020), and they are used to do traffic forecasting. However, those methods are our of the scope of the problem setting of TiaRa because TiaRa aims to augment a discrete-time dynamic graph where the structure of a graph changes over time, not remains static. One promising future research direction is to devise an augmentation method for such traffic networks, which synthetically generates node features over time or forms temporal graph snapshots based on node feature similarities. Another research direction is to expand our method to dynamic signed graphs (Raghavendra et al. 2022) using signed graph diffusion (Jung et al. 2016; Jung, Yoo, and Kang 2022).

C Additional Experiments

Effect of Hyperparameters

We additionally conduct experiments to investigate the effect of hyperparameters of TiaRa. Figure 5 demonstrates the effect of restart probability α and time travel probability β where $0.1 \leq \alpha, \beta \leq 0.5$ in the BitcoinAlpha and the Brain datasets. As shown in the figure, the effect of α and β depends on datasets and models (similar in other datasets). Figure 6 shows the results of link prediction in the RedditBody dataset according to values of γ. Similar to the BitcoinAlpha and the WikiElec dataset, it is crucial to properly adjust γ for the performance.

D Experimental Setting

We describe detailed settings for our experiments including datasets, computing environment, hyperparemters, etc.

Detailed Information of Datasets

We use the following datasets for the temporal link prediction task:

- **BitcoinAlpha**[1]: It is a who-trusts-whom network of people who trade using Bitcoin on platforms called Bitcoin Alpha (Kumar et al. 2018b). We use only edges and their time stamps without rating in this work.
- **WikiElec**[2]: It is a voting network for Wikipedia elections (Leskovec, Huttenlocher, and Kleinberg 2010).
- **RedditBody**[3]: This dataset is a hyperlink network representing connections between two posts, called subreddits, where a sub-reddit is a community on the Reddit (Kumar et al. 2018a).

For the node classification task, we utilize the following datasets[4] used in (Xu et al. 2019):

- **Brain**: This dataset is a network of brain issues where a node represents the tidy cube of brain tissue, and two nodes are connected if they show similar degree of activation. Xu et al. applied PCA to the functional magnetic resonance imaging (fMRI) data to generate initial node features of 20 dimensions.
- **DBLP-3** and **DBLP-5**: These datasets were extracted from the DBLP, and they are co-authorship networks where nodes represent authors. The authors in DBLP-3 and DBLP-5 are from three and five research areas, respectively. Xu et al. used titles and abstracts of papers to produce node features by using word2vec where the features dimension is 100.
- **Reddit**: This is a post network where a node is a post and two posted are connected if they share similar keywords. Xu et al. applied word2vec to comments of a post to extract its node features of 20 dimensions.

Computing Environment and Implementation

We describe detailed specifications of computing environment and implementation used for experiments.

- CPU: Intel(R) Xeon(R) Silver 4215R CPU @ 3.20GHz
- GPU: NVIDIA GeForce RTX 3090 24GB
- CUDA: 11.3, Python: 3.10.4, PyTorch: 1.12.0

Baseline GNN Models

- **GCN** (Kipf and Welling 2017): This is a popular GNN model for a static graph, and based on graph convolutional operations. GCN first symmetrically normalizes an self-looped adjacency matrix $\mathbf{A}$ (i.e., self-loops are added) as $\tilde{\boldsymbol{\mathcal{A}}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ where $\mathbf{D}$ is a diagonal degree matrix of $\mathbf{A}$. After the normalization, it stacks a layer $\sigma(\tilde{\boldsymbol{\mathcal{A}}}\mathbf{X}\mathbf{W})$ given node feature matrix $\mathbf{X}$ and trainable parameter $\mathbf{W}$ where $\sigma(\cdot)$ is a non-linear activation fuction such as ReLU. The hyperparameters are hidden dimension, dropout ratio, and the number of layers.

[1]https://snap.stanford.edu/data/soc-sign-bitcoin-alpha.html
[2]https://snap.stanford.edu/data/wiki-Elec.html
[3]https://snap.stanford.edu/data/soc-RedditHyperlinks.html
[4]https://tinyurl.com/y67ywq6j

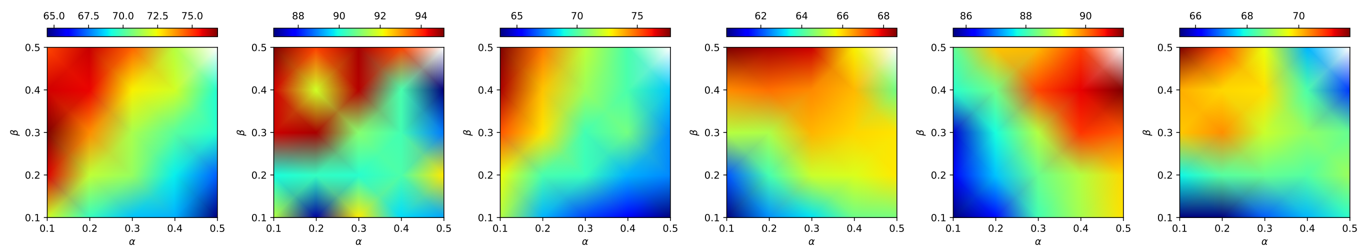
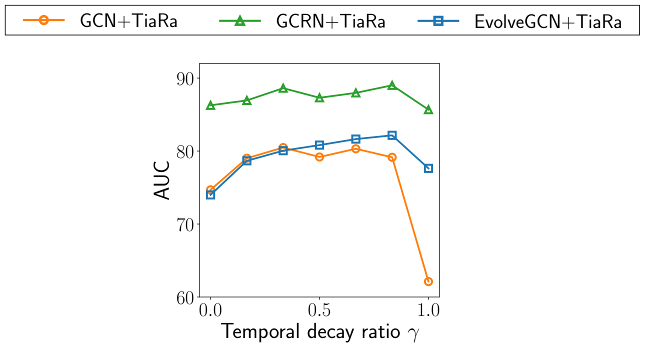

(a) GCN+TIARA on BitcoinAlpha (AUC) (b) GCRN+TIARA on BitcoinAlpha (AUC) (c) EGCN+TIARA on BitcoinAlpha (AUC) (d) GCN+TIARA on Brain (Macro F1) (e) GCRN+TIARA on Brain (Macro F1) (f) EGCN+TIARA on Brain (Macro F1)

Figure 5: Effect of restart probability α and time travel probability β on the BitcoinAlpha and the Brain datasets.



(a) RedditBody

Figure 6: Effect of the temporal decay ratio γ w.r.t link prediction accuracy in the RedditBody dataset.

- **GCRN**[5] (Seo et al. 2018): This combines GNN and RNN to learn dynamic graphs. In general, each operation in the RNN cell is based on matrix multiplications such as $\mathbf{W}_x\mathbf{X}_t$ and $\mathbf{W}_h\mathbf{H}_t$ where $\mathbf{W}_*$ is a parameter, $\mathbf{X}_t$ is an input feature, and $\mathbf{H}_t$ is a hidden state at time t. GCRN extends such matrix multiplications by using graph convolution as $\mathbf{W} *_g \mathbf{X}$ where $*_g$ is a graph convolution operation. Note that Seo et al. employed a GNN based on Chebyshev polynomial (Defferrard, Bresson, and Vandergheynst 2016) for $*_g$ while we use a GCN (Kipf and Welling 2017), described in the above, because the GCN shows better performance, and it is more lightweight than the Chebyshev network. The hyperparameters are hidden dimension, the number of layers, dropout ratio, and the type of RNN (LSTM or GRU).
- **EvolveGCN** (Pareja et al. 2020): It uses an RNN to produce parameters $\mathbf{W}_t$ at each time step t, which is used as GCN parameters while being treated as a hidden state of the RNN. The authors interpret the process as the evolution of GCN parameters over time. The hyperparameters are hidden dimension, the number of layers, dropout ratio, and the type of RNN (LSTM or GRU).

Baseline Augmentation Methods

- **DropEdge**[6] (Rong et al. 2020): It is a drop-based method randomly removing edges at each epoch. The hyperparameter is a dropedge ratio b.t.w. 0 and 1.
- **GDC** (Klicpera, Weißenberger, and Günnemann 2019): It is a diffusion-based method where we use PPR for GDC as our approach is based on random walks. The hyperparameter is α, a restart probability.
- **Merge**: It is a simple baseline merging adjacency matrices from time 1 to t when training a model at time t. The overlapped entries are processed as 1. This method has no hyperparameters.

[5] https://github.com/benedekrozemberczki/pytorch_geometric_temporal

[6] https://github.com/dmlc/dgl

Implementation Details

Details on the temporal link prediction task. The datasets for this task do not contain initial node features; thus, we first preprocess random features $\mathbf{F}_t$ of 32 dimensions for each time t, which are sampled from a standard normal distribution, and use them for all models as initial features. One may use advanced methods (Yoo et al. 2022) for estimating node features when they are partially unavailable in a graph. For each dataset, we aggregate multiple edges within a specific time range, called `time_aggregation`, to represent the data as a discrete-time dynamic graph (i.e., a temporal sequence of graph snapshots), and use $1,200,000$ (seconds) for `time_aggregation`. We also make each graph snapshot undirected since most GNNs require undirected graphs as input although this process is not mandatory for TIARA.

To evaluate this task, we consider existing edges as positive samples, and extract negative edge samples as much as positive ones for each graph snapshot. More specifically, let $d_{t,u}$ be the out-degree of node u in a graph snapshot $\mathcal{G}_t$. Then, for each node u, we randomly sample $d_{t,u}$ nodes, $V_{t,u} = \{v_1, \cdots, v_{d_{t,u}}\}$, from other unconnected nodes except its neighbors, and (u, v_i) s.t. $v_i \in V_{t,u}$ is considered as a negative sample.

As a decoder of the task, we use a simple MLP (i.e., `FC-ReLU-FC`) to predict whether a given pair of nodes forms an edge or not at a specific time step. Given (u, v) and time $t+1$, we concatenate two node embeddings of u and v, which were produced at time t, as the input of the decoder. The decoder outputs scores (or logits) for positive and negative classes, and forwards the scores to the cross entropy loss function. The total loss is the mean of the losses of all samples across all time steps within a training (or validation/test) set. The input dimension of the decoder is 64, and the output dimension is 2. We set the hidden dimension of the decoder to 32. We use the same decoder for all models.

Details on the node classification task. The original dynamic graphs have features, and they are represented as a sequence of graph snapshots. As in (Xu et al. 2019), we randomly split nodes for a train/validation/test set with the 7:1:2 ratio. In this task, as a decoder, we also use a simple MLP having the same architecture as the aforementioned decoder except that this task's decoder receives the embedding of each node on the last time step (note that labels of this task do not have temporal information). The decoder outputs L scores where L is the number of node labels (or classes). We also utilize the cross entropy loss function in this task. The input dimension of the decoder is the feature dimension d of $\mathbf{F}_t$, and the output dimension is L. We set the hidden dimension of the decoder to 32. We exploit the same decoder for all models.

Symmetric trick. As suggested in (Klicpera, Weißenberger, and Günnemann 2019), we also found that making $\tilde{\boldsymbol{\mathcal{X}}}_t$ symmetric can improve predictive performance in several settings, and call this a symmetric trick. Since $\tilde{\boldsymbol{\mathcal{X}}}_t$ is not symmetric, we first make it symmetric as $\hat{\boldsymbol{\mathcal{X}}}_t = (\tilde{\boldsymbol{\mathcal{X}}}_t + \tilde{\boldsymbol{\mathcal{X}}}_t^\top)/2$. We then drop the weights of entries of $\hat{\boldsymbol{\mathcal{X}}}_t$ (i.e., replace nonzero elements to 1), and the resulting matrix is denoted by $\hat{\mathbf{A}}_t$. Note $\hat{\mathbf{A}}_t$ is not normalized. Thus, we perform symmetric normalization on $\hat{\mathbf{A}}_t$, i.e., $\tilde{\boldsymbol{\mathcal{A}}}_t = \hat{\mathbf{D}}_t^{-\frac{1}{2}} \hat{\mathbf{A}}_t \hat{\mathbf{D}}_t^{-\frac{1}{2}}$ where $\hat{\mathbf{D}}_t$ is a diagonal degree matrix of $\hat{\mathbf{A}}_t$. After that, $\tilde{\boldsymbol{\mathcal{A}}}_t$ is fed into a model as an augmented adjacency matrix. We use this technique for graph diffusion-based methods such as GDC and TIARA.

Hyperparameters

We report the hyperparameter search bounds and the used configurations. We summarize the information of hyperparameters that we tuned for experiments in Table 7. COMMON indicates hyperparameters that commonly appear in GNN models or the optimizer. We fix weight decay to 10^{-4}, K of TIARA to 100, and learning rate decay to 0.999 for each epoch. We fix the embedding dimension of GNN models to d which is the feature dimension of $\mathbf{F}_t$. The search bound of each hyperparameters is described in Table 8. We summarize the final hyperparemters obtained by the search in Tables 6∼13.

Table 7: Summary of hyperparameters

Method	**Hyperparameters**
COMMON	learning rate, # of layers, RNN type, dropout ratio
DROPEDGE	dropedge ratio
GDC	restart prob. α, filtering threshold ϵ, symmetric trick
TIARA	restart prob. α, time travel prob. β, filtering threshold ϵ, symmetric trick

Table 8: Search bounds of hyperparameters

Hyperparameters	**Search bounds**
learning rate	$\{0.01, 0.02, 0.05\}$
number of layers	$\{2, 3\}$
RNN type	$\{\text{LSTM}, \text{GRU}\}$
dropout ratio	$[0, 0.5]$ by 0.05
dropedge ratio	$[0.1, 0.9]$ by 0.1
α of GDC	$\{0.01, 0.02, 0.05, 0.1, 0.15, 0.2, 0.3, 0.4, 0.5\}$
α, β of TIARA	$\{(\alpha, \beta)\}$ where $\alpha \in (0,1)$, $\beta \in (0,1)$, and $\alpha + \beta \in (0,1)$
ϵ	$\{0.0001, 0.001, 0.01\}$

Table 6: Searched hyperparameters of GCN in the temporal link prediction task

Augmentation	**Dataset**	**# of layers**	**learing rate**	**RNN type**	**dropout ratio**	**dropedge ratio**	α	β	ϵ	**symmetric trick**
NONE	BitcoinAlpha	3	0.05	-	0	-	-	-	-	-
	WikiElec	3	0.02		0					
	RedditBody	3	0.02		0					
DROPEDGE	BitcoinAlpha	3	0.05	-	0	0.6	-	-	-	-
	WikiElec	3	0.02		0	0.9				
	RedditBody	3	0.02		0	0.9				
GDC	BitcoinAlpha	3	0.05	-	0	-	0.4	-	0.001	Off
	WikiElec	3	0.02		0.5		0.2			
	RedditBody	3	0.02		0.5		0.2			
MERGE	BitcoinAlpha	3	0.05	-	0	-	-	-	-	-
	WikiElec	3	0.05		0					
	RedditBody	2	0.02		0					
TIARA	BitcoinAlpha	3	0.05	-	0	-	0.05	0.2	0.001	Off
	WikiElec	3	0.02		0		0.1	0.2		
	RedditBody	3	0.02		0.5		0.3	0.2		

Table 9: Searched hyperparameters of GCN in the node classification task

Augmentation	Dataset	# of layers	learing rate	RNN type	dropout ratio	dropedge ratio	α	β	ϵ	symmetric trick
NONE	Brain	2	0.01	-	0	-	-	-	-	-
	Reddit	3	0.05		0					
	DBLP3	2	0.05		0					
	DBLP5	2	0.01		0					
DROPEDGE	Brain	2	0.01	-	0.5	0.1	-	-	-	-
	Reddit	3	0.05		0.5	0.8				
	DBLP3	2	0.05		0	0.3				
	DBLP5	2	0.05		0.5	0.6				
GDC	Brain	2	0.01	-	0	-	0.5	-	0.001	Off
	Reddit	3	0.05		0		0.3			
	DBLP3	2	0.05		0		0.3			
	DBLP5	2	0.05		0		0.05			
MERGE	Brain	2	0.02	-	0	-	-	-	-	-
	Reddit	3	0.02		0					
	DBLP3	2	0.01		0					
	DBLP5	2	0.05		0					
TIARA	Brain	2	0.01	-	0	-	0.1	0.8	0.001	Off
	Reddit	3	0.05		0		0.2	0.4		Off
	DBLP3	2	0.05		0.5		0.1	0.4		On
	DBLP5	2	0.01		0.5		0.001	0.009		On

Table 10: Searched hyperparameters of GCRN in the temporal link prediction task

Augmentation	Dataset	# of layers	learing rate	RNN type	dropout ratio	dropedge ratio	α	β	ϵ	symmetric trick
NONE	BitcoinAlpha	3	0.05	LSTM	0	-	-	-	-	-
	WikiElec	2	0.05	LSTM	0					
	RedditBody	3	0.02	LSTM	0					
DROPEDGE	BitcoinAlpha	3	0.05	LSTM	0.5	0.5	-	-	-	-
	WikiElec	2	0.05	LSTM	0.5	0.8				
	RedditBody	3	0.02	LSTM	0	0.7				
GDC	BitcoinAlpha	3	0.05	LSTM	0.5	-	0.2	-	0.001	On
	WikiElec	2	0.05	LSTM	0.5		0.1			
	RedditBody	3	0.02	LSTM	0.5		0.3			
MERGE	BitcoinAlpha	3	0.05	LSTM	0.5	-	-	-	-	-
	WikiElec	3	0.05	LSTM	0.5					
	RedditBody	3	0.01	LSTM	0					
TIARA	BitcoinAlpha	3	0.05	LSTM	0.5	-	0.1	0.3	0.001	On
	WikiElec	2	0.05	LSTM	0.5		0.1	0.3		
	RedditBody	3	0.02	LSTM	0.1		0.1275	0.7225		

Table 11: Searched hyperparameters of GCRN in the node classification task

Augmentation	Dataset	# of layers	learing rate	RNN type	dropout ratio	dropedge ratio	α	β	ϵ	symmetric trick
NONE	Brain	2	0.02	GRU	0	-	-	-	-	-
	Reddit	2	0.01	LSTM	0					
	DBLP3	3	0.02	GRU	0					
	DBLP5	2	0.01	GRU	0					
DROPEDGE	Brain	2	0.02	GRU	0	0.6	-	-	-	-
	Reddit	2	0.05	LSTM	0.5	0.1				
	DBLP3	2	0.02	LSTM	0.5	0.8				
	DBLP5	2	0.01	GRU	0.5	0.8				
GDC	Brain	2	0.01	LSTM	0	-	0.4	-	0.001	Off
	Reddit	2	0.02	LSTM	0		0.1			
	DBLP3	2	0.02	LSTM	0		0.5			
	DBLP5	2	0.02	LSTM	0		0.3			
MERGE	Brain	2	0.02	GRU	0	-	-	-	-	-
	Reddit	2	0.01	LSTM	0					
	DBLP3	2	0.02	LSTM	0					
	DBLP5	2	0.01	GRU	0					
TIARA	Brain	2	0.01	LSTM	0	-	0.5	0.4	0.001	Off
	Reddit	2	0.02	LSTM	0		0.4	0.5		
	DBLP3	2	0.02	LSTM	0		0.4	0.5		
	DBLP5	2	0.02	LSTM	0.5		0.1	0.8		

Table 12: Searched hyperparameters of EvolveGCN (EGCN) in the temporal link prediction task

Augmentation	Dataset	# of layers	learing rate	RNN type	dropout ratio	dropedge ratio	α	β	ϵ	symmetric trick
NONE	BitcoinAlpha	3	0.01	LSTM	0	-	-	-	-	-
	WikiElec	2	0.05	LSTM	0					
	RedditBody	3	0.02	LSTM	0					
DROPEDGE	BitcoinAlpha	3	0.01	LSTM	0	0.8	-	-	-	-
	WikiElec	2	0.05	LSTM	0	0.1				
	RedditBody	3	0.02	LSTM	0	0.5				
GDC	BitcoinAlpha	3	0.01	LSTM	0.5	-	0.2	-	0.001	On
	WikiElec	2	0.05	LSTM	0		0.1			
	RedditBody	3	0.02	LSTM	0.5		0.2			
MERGE	BitcoinAlpha	3	0.05	LSTM	0.5	-	-	-	-	-
	WikiElec	3	0.05	GRU	0					
	RedditBody	3	0.05	LSTM	0					
TIARA	BitcoinAlpha	3	0.01	LSTM	0.5	-	0.1	0.4	0.001	On
	WikiElec	2	0.05	LSTM	0.5		0.3	0.4		
	RedditBody	3	0.02	LSTM	0.5		0.1	0.2		

Table 13: Searched hyperparameters of EvolveGCN (EGCN) in the node classification task

Augmentation	**Dataset**	**# of layers**	**learing rate**	**RNN type**	**dropout ratio**	**dropedge ratio**	α	β	ϵ	**symmetric trick**
NONE	Brain	2	0.01	LSTM	0	-	-	-	-	-
	Reddit	2	0.02	LSTM	0					
	DBLP3	2	0.02	LSTM	0					
	DBLP5	2	0.02	LSTM	0					
DROPEDGE	Brain	2	0.01	LSTM	0	0.6	-	-	-	-
	Reddit	2	0.02	LSTM	0	0.5				
	DBLP3	2	0.02	LSTM	0.5	0.2				
	DBLP5	2	0.02	LSTM	0.5	0.1				
GDC	Brain	2	0.01	LSTM	0	-	0.5	-	0.001	Off
	Reddit	2	0.02	LSTM	0		0.1			
	DBLP3	2	0.02	LSTM	0		0.01			
	DBLP5	2	0.02	LSTM	0		0.5			
MERGE	Brain	2	0.05	LSTM	0	-	-	-	-	-
	Reddit	2	0.01	LSTM	0					
	DBLP3	2	0.01	LSTM	0					
	DBLP5	2	0.02	GRU	0					
TIARA	Brain	2	0.01	LSTM	0	-	0.1	0.5	0.001	Off
	Reddit	2	0.02	LSTM	0.05		0.002	0.008		Off
	DBLP3	2	0.02	LSTM	0.5		0.1	0.5		On
	DBLP5	2	0.02	LSTM	0.7		0.1	0.8		Off