The following publication Y. Yang et al., "Time-Aware Dynamic Graph Embedding for Asynchronous Structural Evolution," in IEEE Transactions on Knowledge and Data Engineering, vol. 35, no. 9, pp. 9656-9670, 1 Sept. 2023 is available at https://dx.doi.org/10.1109/TKDE.2023.3246059.

Time-aware Dynamic Graph Embedding for Asynchronous Structural Evolution

Yu Yang, Hongzhi Yin, Jiannong Cao, *Fellow, IEEE,* Tong Chen, Quoc Viet Hung Nguyen, Xiaofang Zhou, *Fellow, IEEE,* Lei Chen, *Fellow, IEEE*

Abstract—Dynamic graphs refer to graphs whose structure dynamically changes over time. Despite the benefits of learning vertex representations (i.e., embeddings) for dynamic graphs, existing works merely view a dynamic graph as a sequence of changes within the vertex connections, neglecting the crucial asynchronous nature of such dynamics where the evolution of each local structure starts at different times and lasts for various durations. To maintain asynchronous structural evolutions within the graph, we innovatively formulate dynamic graphs as temporal edge sequences associated with joining time of vertices (ToV) and timespan of edges (ToE). Then, a time-aware Transformer is proposed to embed vertices' dynamic connections and ToEs into the learned vertex representations. Meanwhile, we treat each edge sequence as a whole and embed its ToV of the first vertex to further encode the time-sensitive information. Extensive evaluations on several datasets show that our approach outperforms the state-of-the-art in a wide range of graph mining tasks. At the same time, it is very efficient and scalable for embedding large-scale dynamic graphs.

Index Terms—Dynamic graph embedding, Graph evolution, Edge timespan, Graph mining.

1 Introduction

Graphs are one of the most widely used data structures to represent pairwise relations between entities. In many real-world applications, these relations naturally change over time, making the graph dynamic. For example, in an online forum, users can post messages at any time and form a discussion network. Some of them join the discussion by replying or citing the published posts, thus forming edges among existing vertices. Some may choose to unfollow other users or become inactive in the discussion, and this will make them disconnected from other vertices. Since vertices can join, leave and re-join a network at any time, the dynamics of graphs are beneficial for modeling various types of data like traffic, financial transactions, and social media.

Dynamic graph embedding is an effective means to encode the evolution of vertices' connections and properties into vector representations to facilitate downstream applications. In general, to capture the dynamic changes of vertex connections over time, existing approaches represent the dynamic graph as either a snapshot graph sequence (SGS) [1] [2] [3] [4] [5] [6] [7] or neighborhood formation sequences (NFS) [8] [9] [10] [11]. On one hand, SGS segments the graph information into several time slots, where a static snapshot graph is built to represent the node connectivity within each time slot. On the other hand, NFS captures dynamic graph information by using temporal random walk to sample the sequential connections among vertices.

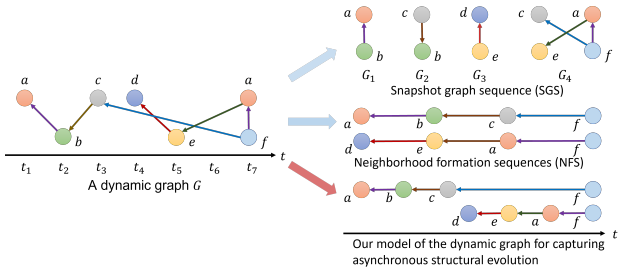


Fig. 1. Modeling a dynamic graph with SGS, NFS, and our approach.

Unfortunately, both SGS and NFS incur inevitable information loss about the dynamics properties of a graph, thus impeding the expressiveness of the eventually learned node embeddings. In Fig. 1, we provide an example on how SGS and NFS represent the dynamic graph G, respectively. For SGS, a snapshot graph is built for each of the 4 time slots ($[t_3, t_4]$ and $[t_5, t_6]$ are omitted as there are no new connections). Obviously, SGS tends to suffer from sparsity issues with a fine-grained time granularity, e.g., apart from G_4, the vertices and edge in G_1, G_2, and G_3 are too sparse for effectively learning node embeddings. Should these three snapshot graphs be merged to alleviate the sparsity, the subtle structural evolutions at different time steps will be neglected. Furthermore, a largely overlooked fact in dynamic graphs is that, the structural evolutions are **asynchronous**, i.e., the exact time consumed by each edge update varies in different cases. For example, on the left of Fig. 1, the connections among vertices $\{a, b, c, f\}$ and $\{d, e, a, f\}$, evolve asynchronously since the edges have different starting times and durations. As a result, modeling G as a SGS shown in Fig. 1 will interrupt the continuous

- *Y. Yang and J. Cao are with the Department of Computing, The Hong Kong Polytechnic University, China. E-mail: {cs-yu.yang, jiannong.cao}@polyu.edu.hk*
- *H. Yin and T. Chen are with the School of Information Technology and Electrical Engineering, The University of Queensland, Australia. E-mail: {tong.chen, h.yin1}@uq.edu.au*
- *Q. V. H. Nguyen is with The Griffith University, Australia. E-mail: quocviethung.nguyen@griffith.edu.au*
- *X. Zhou and L. Chen are with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, China. E-mail: {zxf, leichen}@cse.ust.hk*

evolution of vertices $\{a, b, c, f\}$ with an irrelevant snapshot G_3. Therefore, incorrect graph dynamics will be embedded, as such asynchronous temporal information is totally lost when treating a dynamic graph as an array of static snapshots.

Compared with SGS, NFS is more robust to sparsity as a certain amount of substructures of the dynamic graph are considered for different time steps owing to the temporal random walk strategy. However, it still fails to capture the graph's asynchronous structural evolutions. Taking G in Fig. 1 as an example, the local structure among $\{a, b, c, f\}$ and $\{d, e, a, f\}$ starts evolving at t_1 and t_4, respectively. The former structure takes 6 time steps to completely link four vertices, which is slower than the latter one that takes just 3 steps. Nevertheless, as shown in the middle-right part of Fig. 1, the NFS model neglects this important temporal discrepancy, thus being unable to preserve such asynchronous structural evolutions.

In this paper, we study the important yet overlooked problem in dynamic graph embedding, namely fully capturing the asynchronous structural evolutions of a graph. Such asynchronous nature can be interpreted as the starting time and duration of a new edge that emerges between two vertices, and is of great importance to a wide range of time-sensitive applications, e.g., online information propagation modeling and crime prediction. However, to achieve the goal, we are confronted with three major challenges. (1) Capturing the asynchronous structural evolutions in a dynamic graph. The representation of a dynamic graph should capture not only the dynamic connections among vertices but also the starting time and duration of such evolutions, so as to provide sufficient knowledge about the asynchronous characteristics of a dynamic graph. (2) Embedding spatial and temporal dynamics of edges. When vertices join, leave, and re-join the graph, the edges they formed will change accordingly, which brings spatial dynamics. Meanwhile, it takes a different amount of time for vertices to establish new links, leading to dramatic temporal dynamics. To preserve the dynamic edge formation, both the spatial and temporal dynamics should be fully encoded in the dynamic node embeddings. (3) Preserving asynchronous evolutions of local structures. Some local structures evolve early, while others evolve later. The embedding algorithm should account for the evolution starting time for every local structure along with its dynamic edges, such that the patterns within asynchronous structural evolutions can be effectively learned to facilitate predictions into the future.

In light of these challenges, we propose to represent a dynamic graph as a set of temporal edges, coupled with the respective joining time of vertices (ToV) and timespan of edges (ToE) as two crucial indicators of the asynchronous properties. A time-centrality-biased temporal random walk is then developed to sample the local structures as temporal edge sequences. The ToV of the first vertex in a temporal edge sequence corresponds to the evolution starting time of the local structure, and the total ToE indicates how long the evolution of the current local structure takes. Hence, the temporal edge sequences successfully capture the asynchronous structural evolutions for learning expressive node embeddings. Moreover, we propose a novel **T**ime-**A**ware **D**ynamic **G**raph **E**mbedding (TADGE) algorithm to embed the asynchronous structural evolutions into vertex representations. In order to thoroughly incorporate both the spatial and temporal dynamics of the edge formation, we design a time-aware Transformer model. Intuitively, a vertex forms a new edge by linking another vertex that has high affinity with it. Therefore, we build a Transformer [12] to embed vertices' dynamic connections as their self-attentive embeddings through an encoder-decoder framework, making the learned embeddings for every vertex carry the structural information and ToE from its connected neighbors. In order to preserve the asynchronous evolution of local structures, we learn an overall representation of every edge sequence by accounting for its evolution starting time. Lastly, we fuse the vertex- and sequence-level representations to generate the final embedding for each vertex, encoding the patterns of its asynchronous structural evolutions in the dynamic graph to support downstream tasks.

Our contributions are highlighted as follows:

- **A New Problem.** To the best of our knowledge, we are the first to study the problem of embedding the asynchronous structural evolutions of a dynamic graph, in which the evolution starting time and duration of each local structures vary significantly.
- **A Novel Representation of Dynamic Graphs.** We propose a time-centrality-biased temporal random walk to innovatively formulate the dynamic graph as temporal edge sequences associated with ToV and ToE tags, which preserves the asynchronous structural evolutions for learning vertex embeddings.
- **A New Approach for Dynamic Graph Embedding.** We propose TADGE, which is a novel time-aware graph embedding approach that learns expressive dynamic vertex embeddings by fusing information of both the dynamic edge formation and the asynchronous local structure evolutions.
- **Extensive Experiments.** We conduct experiments on several large-scale public datasets on dynamic graphs. Experimental results demonstrate the superiority of TADGE, which outperforms the state-of-the-arts and also shows significant advances in training efficiency and scalability.

The remainder of this paper is organized as follows. We present the problem definition in Section 2, followed by our proposed time-centrality-biased temporal random walk and TADGE algorithm in Sections 3 and 4. Experimental results are reported in Section 5, followed by the literature review in Section 6 before we conclude the paper.

2 Problem Definition

In this section, we give the definition of a dynamic graph and formulate the problem of dynamic graph embedding for preserving the asynchronous structural evolutions.

In a dynamic graph G, vertices join the graph at any time as either isolated vertices or forming edges with existing ones. Thus, we denote a vertex v_i joining G at time t as v_i^t where t is the joining time of the vertex (ToV) and $i = 1, 2, \cdots, n$. Since vertices join, leave and rejoin the graph dynamically, which triggers the structural evolution, we denote all ToVs of v_i as its ToV set T_{v_i} in which the number

of appearances of v_i in G is $|T_{v_i}|$. $|\cdot|$ is an operator for counting the number of elements in a set. When v_i^t links to an existing vertex $v_j^{t'}$ whose ToV $t' < t$ and $j = 1, 2, \cdots, n$, they form a temporal edge $e_{v_i,v_j}^{t,\delta} = (v_i, v_j, t, \delta, w)$ at time t, where $\delta = t - t'$ is the timespan of the edge (ToE) indicating how long it takes for v_i^t to form an edge with $v_j^{t'}$, and w is the edge weight. If $i = j$, a self-link is formed.

Next, we give a formal definition of a dynamic graph that is composed of the dynamic appearing of vertices at different times with the edges they formed, and then we formulate the problem of dynamic graph embedding for preserving asynchronous structural evolutions.

Definition 1 ***Dynamic Graph.*** *A dynamic graph $G = \{V, E, T_V\}$, where $V = \{v_1, v_2, \cdots\}$ denotes a vertex set containing $|V|$ vertices in G, and $T_V = \{T_{v_1}, T_{v_2}, \cdots\}$ is the ToV set of every vertex in V. E is the temporal edge set in which $e_{v_i,v_j}^{t,\delta} = (v_i, v_j, t, \delta, w) \in E$ is a temporal edge linking $v_i, v_j \in V$. $t \in T_{v_i}$ is the ToV of an upcoming vertex v_i and indicates the edge forming time. $t' \in T_{v_j}$ is the ToV of an existing vertex v_j and $t' < t$. $\delta = t - t'$ is the ToE. w is the edge weight.*

Definition 2 ***Dynamic Graph Embedding.*** *Given a dynamic graph $G = \{V, E, T_V\}$, the objective is to learn a mapping function $f : v \mapsto r_v \in \mathbb{R}^k$ for $\forall v \in V$ such that the representation r_v preserves the asynchronous structural evolution of v in terms of asynchronous evolution starting time and duration, where k is a positive integer indicating the dimension of r_v.*

Since we aim to embed the asynchronous structural evolutions in a dynamic graph, we assume $|V|$ is well-known and mainly focus on the edge updates in our study. Meanwhile, our approach is compatible with any inductive learning schemes to handle newly joined vertices.

3 Capturing Asynchronous Structural Evolutions in the Dynamic Graph

Directly learning embeddings from the origin graph G is difficult due to its complexity and dynamics. Thus, it is necessary to transform the original dynamic graph into a proper format that captures its dynamic structural evolution and is easy for later embedding. However, regardless of modeling the dynamic graph as either SGS or NFS, the asynchronous structural evolutions cannot be captured fairly well. Inspired by [8] and [13], we propose a time-centrality-biased temporal random walk to sample the dynamic graph and model it as a set of temporal edge sequences that properly capture the asynchronous structural evolutions.

Specifically, we first randomly select a set of initial vertices via uniform distribution, which appear at m different times and satisfy $|V| \leq m < \sum_{v \in V} |T_v|$. Since there are $|V|$ vertices totally appearing $\sum_{v \in V} |T_v|$ times in G, each of vertices' occurrence has the same probability $p = \frac{1}{\sum_{v \in V} |T_v|}$ to be selected as the initial one. This makes the sequences sampled by the following temporal random walk cover the entire graph evenly.

Next, we perform the temporal random walk to sample the connected vertices starting from every initial vertex. The walker will only visit vertices whose ToV is greater than the previous one, making the sampled sequences be in line with the structural evolutions of G. Usually, the upcoming vertices have higher chances to form edges with the high centrality ones [14]. As the events/relationships happen in recent times may have stronger influence to the current vertex comparing to that of happened long time ago, we heavily sample vertices that are close in time to the current one. In other words, the walker prefers to visit a vertex through the edge with a smaller ToE but a higher centrality. Therefore, given a vertex v^t, our temporal random walk samples the next vertex $v_{next}^{t'}$ from its neighborhood set Γ_{v^t} based on a transition probability by Eq. (1) following a time-centrality-biased distribution, where the ToV of every vertex in Γ_{v^t} is greater than t, $degree(\cdot)$ is the vertex's degree for measuring its centrality, δ_{v,v^t} is the ToE of the edge connecting two vertices v and v^t.

$$p_{rw}(v_{next}^{t'}) = \frac{degree(v_{next}^{t'})}{\sum_{v \in \Gamma_{v^t}} degree(v)} \left(1 - \frac{\delta_{v_{next}^{t'}, v^t}}{\sum_{v \in \Gamma_{v^t}} \delta_{v,v^t}}\right) \quad (1)$$

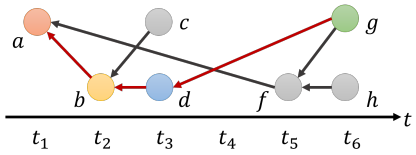


Fig. 2. An example of the time-centrality-biased temporal random walk starting from a vertex a. Since vertices b and f have the same degrees but ToE $\delta_{a,b} < \delta_{a,f}$, the walker will prefer b more than f. Although vertices c and d have the same ToE, the degree of d is higher than that of c, resulting in higher probability to be sampled by the walker after b.

Fig. 2 shows a running example of the time-centrality-biased temporal random walk. Leveraging the centrality of vertices and ToE to sample the dynamic graph makes the vertex distribution and its linkage evolutionary patterns in the generated corpus consistent with the original graph, thus providing comprehensive information for later embedding.

We record the first ℓ vertices that the walker visits to construct the edge sequences s_t, $t = 1, 2, \cdots, m$. The ToV of the first vertex in s_t exactly is the evolution starting time and the total sum of ToE of all edges in s_t indicates the time the evolutions of s_t lasts. Finally, the dynamic graph G has been modeled as a set of edge sequence $\hat{G} = \{s_1, s_2, \cdots, s_t, \cdots, s_m\}$. Every edge sequence s_t is a local structure consisting of ℓ vertices in the dynamic graph. An example is shown on the bottom-right in Fig. 1 demonstrating that the time-centrality-biased temporal random walk is capable of capturing the asynchronous structural evolutions in the dynamic graph.

In order to easily facilitate the embedding algorithm, we further attach a virtual vertex $\langle EOS \rangle$ with zero ToE to the end of every s_t. If the walker early stops before reaching the maximum sampling number, we supplement $\langle EOS \rangle$ at the end to make the sequence have the same length as others.

4 Embedding Asynchronous Structural Evolutions in The Dynamic Graph

In this section, we present the details of our proposed TADGE to embed the asynchronous structural evolutions from $\hat{G} = \{s_1, s_2, \cdots, s_t, \cdots, s_m\}$. TADGE consists of three parts: **(1) Edge Formation Embedding:** a Time-aware Transformer to embed the dynamic connection changes among

vertices while preserving the ToE; **(2) Structural Evolution Embedding:** a multi-head self-attention model to embed the asynchronous evolution starting time for every local structure in the dynamic graph; and **(3) Representation Fusion:** encoding final vertex representation by fusing the above edge formation and structural evolution embeddings.

In sections 4.1 to 4.3, we first introduce how TADGE encodes the representation for a vertex v_i in one of its occurring edge sequences $s_t \in \hat{G}$. Then, in section 4.4, we present the strategies for training the TADGE by using all edge sequences containing v_i to update its representation.

In the rest of this paper, we use bold uppercase letters to denote matrices, bold lowercase letters to denote vectors, and non-bold characters to denote scalars if not clarified.

4.1 Embedding Dynamic Edge Formation with ToE

When a vertex comes and forms a new edge, it actually is selecting an existing vertex in the graph to connect based on past connection evolution and ToE. From the vertex point of view, the edge sequences carrying vertices' connection evolution can be regarded as the sequences of vertices linking by these edges. Since the edges have different ToE, the time interval between consecutive vertices in the sequence varies from each other, thus bringing time-varying sequential dependency to vertices in the edge sequence. Therefore, we embed the dynamic edge formation by learning (1) the time-varying sequential dependency among vertices, and (2) pairwise connections among vertices.

Inspired by the R-Transformer [15], we propose a Time-aware Transformer consisting of a t-LSTM model and a Transformer to respectively learn vertices' sequential dependency with ToE and their pairwise connections in every edge sequence s_t. The overall architecture is shown in Fig. 3 and $\hat{\boldsymbol{r}}_{v_i}$ is the obtained edge formation embedding for v_i.

4.1.1 Learning Time-varying Sequential Dependency

Let us denote the target vertex that we are going to embed as v_i. To learn the sequential impact of ahead connected vertices and the ToE to v_i in s_t, we build a t-LSTM model that starts from a standard LSTM unit and connects to time-aware LSTM units [16]. It discounts the short-term effects from the long-term memory and supplements the impact of ToE between two consecutive nodes in the sequence, thus learning the time-varying sequential dependency.

Specifically, given the input information $\boldsymbol{x}_{v_i} \in \mathbb{R}^k$ of v_i and its connected vertex v_j's overall memory $\boldsymbol{c}_{v_j}$ and the hidden state $\boldsymbol{h}_{v_j} \in \mathbb{R}^k$, the time-aware LSTM unit first decomposes $\boldsymbol{c}_{v_j}$ into short-term memory $\boldsymbol{c}^S_{v_j}$ and long-term memory $\boldsymbol{c}^L_{v_j}$ as below:

$$\boldsymbol{c}^S_{v_j} = \tanh\left(\boldsymbol{W}_d \boldsymbol{c}_{v_j} + \boldsymbol{b}_d\right) \tag{2}$$

$$\boldsymbol{c}^L_{v_j} = \boldsymbol{c}_{v_j} - \boldsymbol{c}^S_{v_j} \tag{3}$$

where $\{\boldsymbol{W}_d, \boldsymbol{b}_d\}$ are the trainable weights and bias for the subspace decomposition. Next, we further decay the decayed short-term memory $\boldsymbol{c}^S_{v_j}$ by a heuristic function in Eq. (4) such that the longer ToE the fewer effects to the short-term memory. δ_{v_i,v_j} is the ToE of the edge linking v_i and v_j while e is the Euler's number.

$$\hat{\boldsymbol{c}}^S_{v_j} = \frac{\boldsymbol{c}^S_{v_j}}{\log(e + \delta_{v_i,v_j})} \tag{4}$$

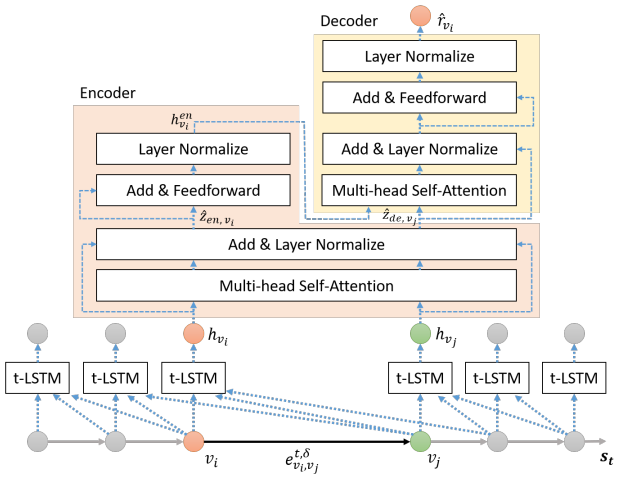


Fig. 3. An illustration of the Time-aware Transformer for embedding v_i.

Lastly, we compose the adjusted overall memory back by

$$\tilde{\boldsymbol{c}}_{v_j} = \boldsymbol{c}^L_{v_j} + \hat{\boldsymbol{c}}^S_{v_j} \tag{5}$$

The rest parts in the time-aware LSTM unit are the same as the standard LSTM as shown below:

$$\boldsymbol{f}_{v_i} = \text{sigmoid}(\boldsymbol{W}_\text{f}\boldsymbol{x}_{\text{v}_\text{i}} + \boldsymbol{U}_\text{f}\boldsymbol{h}_{\text{v}_\text{j}} + \boldsymbol{b}_\text{f}) \tag{6}$$

$$\boldsymbol{g}_{v_i} = \text{sigmoid}(\boldsymbol{W}_\text{g}\boldsymbol{x}_{\text{v}_\text{i}} + \boldsymbol{U}_\text{g}\boldsymbol{h}_{\text{v}_\text{j}} + \boldsymbol{b}_\text{g}) \tag{7}$$

$$\boldsymbol{o}_{v_i} = \text{sigmoid}(\boldsymbol{W}_\text{o}\boldsymbol{x}_{\text{v}_\text{i}} + \boldsymbol{U}_\text{o}\boldsymbol{h}_{\text{v}_\text{j}} + \boldsymbol{b}_\text{o}) \tag{8}$$

$$\bar{\boldsymbol{c}}_{v_i} = \text{sigmoid}(\boldsymbol{W}_\text{c}\boldsymbol{x}_{\text{v}_\text{i}} + \boldsymbol{U}_\text{c}\boldsymbol{h}_{\text{v}_\text{j}} + \boldsymbol{b}_\text{c}) \tag{9}$$

$$\boldsymbol{c}_{v_i} = \boldsymbol{f}_{v_i} * \tilde{\boldsymbol{c}}_{v_j} + \boldsymbol{g}_{v_i} * \bar{\boldsymbol{c}}_{v_i} \tag{10}$$

$$\boldsymbol{h}_{v_i} = \boldsymbol{o}_{v_i} * \tanh(\boldsymbol{c}_{v_i}) \tag{11}$$

$\boldsymbol{c}_{v_i}$ is v_i's current memory. $\boldsymbol{h}_{v_i}$ is the output hidden state of v_i carrying the time-varying sequential dependency between itself and vertices in s_t. $*$ is the element-wise multiplication. $\{\boldsymbol{W}_f, \boldsymbol{U}_f, \boldsymbol{b}_f\}$, $\{\boldsymbol{W}_g, \boldsymbol{U}_g, \boldsymbol{b}_g\}$, $\{\boldsymbol{W}_o, \boldsymbol{U}_o, \boldsymbol{b}_o\}$, and $\{\boldsymbol{W}_c, \boldsymbol{U}_c, \boldsymbol{b}_c\}$ are the trainable weights and bias of the forget gate f, input gate g, output gate o, and candidate memory $\bar{c}$. Consequently, our t-LSTM model is capable to learn the impact of past connections and ToE on the target vertex.

4.1.2 Embedding Pairwise Connection between Vertices

On top of the t-LSTM, we embed the direct connection between v_i and v_j by the self-attention mechanism in a Transformer architecture [12]. Intuitively, the edge formation process is to select an existing vertex from the graph for the upcoming one v_i to link to. In another word, v_i is able to view the candidate vertices in the graph and then determines which one it will form the edge with. We model this process by the self-attention mechanism that contains two steps. First, we build an encoder to compute the self-attention between v_i and vertices in the local structure s_t that v_i belongs. It measures the impact of connected vertices in the local structure for v_i to form edges. Second, regarding the encoder output as the context information of forming the edges, we further build a decoder to learn which vertices v_i exactly connects to, thus well embedding the dynamic edge formation.

Encoder. Given the t-LSTM embeddings of ℓ vertices in the edge sequence s_t, denoting as $\boldsymbol{H}_v =$

$[\boldsymbol{h}_{v_1}, \boldsymbol{h}_{v_2}, \cdots, \boldsymbol{h}_{v_\ell}]^T \in \mathbb{R}^{\ell \times k}$, we start with the multi-head self-attention module:

$$\boldsymbol{Z}^o_{en} = \text{softmax}\left(\frac{\boldsymbol{Q}^o_{en}\boldsymbol{K}^{o\ T}_{en}}{\sqrt{k}} + \boldsymbol{M}\right)\boldsymbol{V}^o_{en} \tag{12}$$

where $\boldsymbol{Z}^o_{en} \in \mathbb{R}^{\ell \times k}$ is the computed attention from the attention-head $o = 1, 2, \cdots, \bar{o}$ while $\sqrt{k}$ is a scaling factor to smooth the softmax calculation for avoiding extremely large values of the inner product. $\boldsymbol{Q}^o_{en}, \boldsymbol{K}^o_{en}, \boldsymbol{V}^o_{en} \in \mathbb{R}^{\ell \times k}$ respectively represent the queries, keys, and values of the self-attention obtained by the linear projection of $\boldsymbol{H}_v$,

$$\boldsymbol{Q}^o_{en} = \boldsymbol{H}_v\boldsymbol{W}^{Q,o}_{en}, \boldsymbol{K}^o_{en} = \boldsymbol{H}_v\boldsymbol{W}^{K,o}_{en}, \boldsymbol{V}^o_{en} = \boldsymbol{H}_v\boldsymbol{W}^{V,o}_{en}, \tag{13}$$

where $\boldsymbol{W}^{Q,o}_{en}, \boldsymbol{W}^{K,o}_{en}, \boldsymbol{W}^{V,o}_{en} \in \mathbb{R}^{k \times k}$ are trainable projection weights for the queries, keys, and values at every attention-head. $\boldsymbol{M} \in \{0, -\infty\}^{\ell \times \ell}$ is a constant attention mask. Since the vertex v_i cannot see and form edges with v_j that joins s_t after v_i, we set $M_{i,j} = -\infty$ if $i < j$. Therefore, the attention value between v_i and not-yet-appeared vertex v_j will be 0 after softmax calculation in Eq. (12). For the vertex v_j that joins s_t earlier than v_i, i.e., $i \geq j$, we disable the mask by setting $M_{i,j} = 0$ and compute their attentions. Unlike GAT [5] which computes attention between directly connected vertices and merely preserves the first-order proximity of the graph structure, we learn the attention between the target vertex and other vertices in the local structure that the target one belongs. Hence, our attention mechanism is capable of preserving both first-order and higher-order proximity of the graph structure.

Next, we concatenate all attentions obtained from every attention-head, denoting as $\boldsymbol{Z}_{en}$, and propagate low-layer queries to $\boldsymbol{Z}_{en}$ in residual connections with the layer normalization as shown in Eq. (16), which can improve the expressive capability and prevent from the vanishing gradient during training.

$$\boldsymbol{Z}_{en} = [\boldsymbol{Z}^1_{en}, \boldsymbol{Z}^2_{en}, \cdots, \boldsymbol{Z}^{\bar{o}}_{en}] \in \mathbb{R}^{\ell \times \bar{o}k} \tag{14}$$

$$\boldsymbol{Q}_{en} = [\boldsymbol{Q}^1_{en}, \boldsymbol{Q}^2_{en}, \cdots, \boldsymbol{Q}^{\bar{o}}_{en}] \in \mathbb{R}^{\ell \times \bar{o}k} \tag{15}$$

$$\hat{\boldsymbol{Z}}_{en} = LN(\boldsymbol{Z}_{en} + \boldsymbol{Q}_{en}) = \boldsymbol{S}_{ln} * \frac{\boldsymbol{Z}_{en} + \boldsymbol{Q}_{en} - \mu}{\epsilon} + \boldsymbol{B}_{ln} \tag{16}$$

$LN(\cdot)$ is the layer normalization function where μ and ϵ are the mean and variance of elements in the input tensor, and $\boldsymbol{S}_{ln}$ and $\boldsymbol{B}_{ln}$ are trainable scaling weights and bias for maintaining the representation ability of the network. $*$ is the element-wise multiplication.

Lastly, we fuse the multi-head attentions $\hat{\boldsymbol{Z}}_{en}$ by a two-layer fully connected feed-forward network (FFN) with a non-linear activation $\text{ReLU}(\cdot) = \max\{0, \cdot\}$ under the residual connection setting and following by the layer normalization to obtain the attention embedding $\boldsymbol{H}^{en}_v$:

$$\boldsymbol{H}^{en}_v = LN(\text{ReLU}(\hat{\boldsymbol{Z}}_{en}\boldsymbol{W}^1_f + \boldsymbol{B}^1_f + \hat{\boldsymbol{Z}}_{en})\boldsymbol{W}^2_f + \boldsymbol{B}^2_f) \tag{17}$$

where $\boldsymbol{W}^1_f \in \mathbb{R}^{\bar{o}k \times \bar{o}k}$, $\boldsymbol{W}^2_f \in \mathbb{R}^{\bar{o}k \times k}$, $\boldsymbol{B}^1_f \in \mathbb{R}^{\ell \times \bar{o}k}$ and $\boldsymbol{B}^2_f \in \mathbb{R}^{\ell \times k}$ are the trainable weights and bias of the FFN. Each row in $\boldsymbol{H}^{en}_v$ corresponds to the encoding of a vertex in s_t.

Decoder. The goal of a decoder is to learn which vertices the encoding one connects to so that an edge forms. Thus, the input of decoder is the t-LSTM embeddings of the second vertex to the last one in s_t, denoting as $\boldsymbol{H}'_v = [\boldsymbol{h}_{v_2}, \boldsymbol{h}_{v_3}, \cdots, \boldsymbol{h}_{v_\ell}, \boldsymbol{h}_{EOS}]^T \in \mathbb{R}^{\ell \times k}$, and their attentions $\hat{\boldsymbol{Z}}'_{en}$ obtained by the encoder through Eq. (12) to (16).

To further learn the pair-wise connection in s_t, we build another self-attention module, employing $\hat{\boldsymbol{Z}}'_{en}$ as queries, $\boldsymbol{H}^{en}_v$, which is the self-attentive embeddings of source vertices, as keys K^o_{de} and values V^o_{de} as showed in Eq. (18),

$$\boldsymbol{Q}^o_{de} = \hat{\boldsymbol{Z}}'_{en}\boldsymbol{W}^{Q,o}_{de}, \boldsymbol{K}^o_{de} = \boldsymbol{H}^{en}_v\boldsymbol{W}^{K,o}_{de}, \boldsymbol{V}^o_{de} = \boldsymbol{H}^{en}_v\boldsymbol{W}^{V,o}_{de} \tag{18}$$

where $\boldsymbol{W}^{Q,o}_{de} \in \mathbb{R}^{\bar{o}k \times k}$ and $\boldsymbol{W}^{K,o}_{de}, \boldsymbol{W}^{V,o}_{de} \in \mathbb{R}^{k \times k}$ are trainable projection weight matrices at every attention-head $o = 1, 2, \cdots, \bar{o}$. Then, we calculate their self-attention by:

$$\boldsymbol{Z}^o_{de} = \text{softmax}\left(\frac{\boldsymbol{Q}^o_{de}\boldsymbol{K}^{o\ T}_{de}}{\sqrt{k}} + \boldsymbol{M}\right)\boldsymbol{V}^o_{de} \tag{19}$$

$$\boldsymbol{Z}_{de} = [\boldsymbol{Z}^1_{de}, \boldsymbol{Z}^2_{de}, \cdots, \boldsymbol{Z}^{\bar{o}}_{de}] \in \mathbb{R}^{\ell \times \bar{o}k} \tag{20}$$

$$\boldsymbol{Q}_{de} = [\boldsymbol{Q}^1_{de}, \boldsymbol{Q}^2_{de}, \cdots, \boldsymbol{Q}^{\bar{o}}_{de}] \in \mathbb{R}^{\ell \times \bar{o}k} \tag{21}$$

$$\hat{\boldsymbol{Z}}_{de} = LN(\boldsymbol{Z}_{de} + \boldsymbol{Q}_{de}) \tag{22}$$

Lastly, a two-layer fully connected FFN with ReLU activation is built to obtain the edge formation embedding $\hat{\boldsymbol{R}}_v$:

$$\hat{\boldsymbol{R}}_v = LN(\text{ReLU}(\hat{\boldsymbol{Z}}_{de}\boldsymbol{W}^3_f + \boldsymbol{B}^3_f + \hat{\boldsymbol{Z}}_{de})\boldsymbol{W}^4_f + \boldsymbol{B}^4_f) \tag{23}$$

where $\boldsymbol{W}^3_f \in \mathbb{R}^{\bar{o}k \times \bar{o}k}$, $\boldsymbol{W}^4_f \in \mathbb{R}^{\bar{o}k \times k}$, $\boldsymbol{B}^3_f \in \mathbb{R}^{\ell \times \bar{o}k}$ and $\boldsymbol{B}^4_f \in \mathbb{R}^{\ell \times k}$ are trainable weights and bias of this FFN. Each row in $\hat{\boldsymbol{R}}_v$ corresponds to the edge formation embedding of a vertex in s_t.

Since the vertices' sequential dependency and ToE have been well preserved by the t-LSTM module and the encoder-decoder merely needs to learn the pairwise connection between vertices in s_t, it is not necessary to incorporate with the position embedding which is used in the original Transformer. In order to further boost the representation power of our Time-aware Transformer model, we respectively stack N blocks of the multi-head attention in both encoder and decoder. We adopt dropout on every residual FFN as a regularization strategy to prevent the model from overfitting. Eventually, our Time-aware Transformer model is capable of simultaneously embedding the edges connecting pairs of vertices together with their ToE.

4.2 Structure Embedding with Evolution Starting Time

When we treat every edge sequence s_t as a whole, the dynamic graph becomes a sequence $\hat{G} = \{s_1, \cdots, s_t, \cdots, s_m\}$ representing the evolving time series of the local structures. This new sequence carries the global evolution patterns of G that consists of the asynchronous evolving of every local structure. We are going to learn a representation vector $\boldsymbol{r}_{s_t}$ for the local structure s_t, where $t = 1, 2, \cdots, m$. The structure embedding $\boldsymbol{r}_{s_t}$ should preserve the sequential dependency of every local structure in $\hat{G}$ and its evolution starting time which exactly is the ToV of its first vertex.

Let's denote the edge formation embedding of a vertex v_i as $\hat{\boldsymbol{r}}_{v_i}$ that corresponds to a row vector in $\hat{\boldsymbol{R}}_v$. We first aggregate $\hat{\boldsymbol{r}}_{v_i}$ for every v_i in s_t to get the initial structure embedding $\boldsymbol{h}_{s_t}$ as shown in Eq. (24). Since the virtual vertex $\langle EOS \rangle$ appears at the end of every s_t and does not contribute anything to the structures, we merely count the first ℓ vertices in s_t to get $\boldsymbol{h}_{s_t}$ regardless they are $\langle EOS \rangle$ or not.

For $\forall s_t \in \hat{G}$, the initial structure embeddings can be written into a matrix form $\boldsymbol{H}_s = [\boldsymbol{h}_{s_1}, \boldsymbol{h}_{s_2}, \cdots, \boldsymbol{h}_{s_m}]^T \in \mathbb{R}^{m \times k}$.

$$\boldsymbol{h}_{s_t} = \sum_{v_i \in s_t} \hat{\boldsymbol{r}}_{v_i}. \tag{24}$$

We again employ the self-attention to learn the sequential relationship among local structures as shown below:

$$\boldsymbol{Q}_s^o = \boldsymbol{H}_s \boldsymbol{W}_s^{Q,o}, \boldsymbol{K}_s^o = \boldsymbol{H}_s \boldsymbol{W}_s^{K,o}, \boldsymbol{V}_s^o = \boldsymbol{H}_s \boldsymbol{W}_s^{V,o} \tag{25}$$

$$\boldsymbol{Z}_s^o = \text{softmax}\left(\frac{\boldsymbol{Q}_s^o \boldsymbol{K}_s^{oT}}{\sqrt{k}}\right) \boldsymbol{V}_s^o \tag{26}$$

$$\boldsymbol{Z}_s = [\boldsymbol{Z}_s^1, \boldsymbol{Z}_s^2, \cdots, \boldsymbol{Z}_s^{\bar{o}}] \in \mathbb{R}^{m \times \bar{o}k} \tag{27}$$

$$\boldsymbol{Q}_s = [\boldsymbol{Q}_s^1, \boldsymbol{Q}_s^2, \cdots, \boldsymbol{Q}_s^{\bar{o}}] \in \mathbb{R}^{m \times \bar{o}k} \tag{28}$$

$$\hat{\boldsymbol{Z}}_s = LN(\boldsymbol{Z}_s + \boldsymbol{Q}_s) \tag{29}$$

where $\boldsymbol{Q}_s^o, \boldsymbol{K}_s^o, \boldsymbol{V}_s^o \in \mathbb{R}^{m \times k}$ respectively represent the queries, keys, and values obtained by the linear projection of $\boldsymbol{H}_s$. $\boldsymbol{W}_s^{Q,o}, \boldsymbol{W}_s^{K,o}, \boldsymbol{W}_s^{V,o} \in \mathbb{R}^{k \times k}$ are trainable projection weight matrices at every attention-head $o = 1, 2, \cdots, \bar{o}$. We again build a two-layer fully connected FFN with the ReLU activation to obtain the structure embedding $\hat{\boldsymbol{R}}_s$:

$$\hat{\boldsymbol{R}}_s = LN(\text{ReLU}(\hat{\boldsymbol{Z}}_s \boldsymbol{W}_f^5 + \boldsymbol{B}_f^5 + \hat{\boldsymbol{Z}}_s)\boldsymbol{W}_f^6 + \boldsymbol{B}_f^6) \tag{30}$$

where $\boldsymbol{W}_f^5 \in \mathbb{R}^{\bar{o}k \times \bar{o}k}$, $\boldsymbol{W}_f^6 \in \mathbb{R}^{\bar{o}k \times k}$, $\boldsymbol{B}_f^5 \in \mathbb{R}^{m \times \bar{o}k}$ and $\boldsymbol{B}_f^6 \in \mathbb{R}^{m \times k}$ are the trainable weights and bias of the FFN.

To this end, we obtain the structure embedding $\hat{\boldsymbol{r}}_{s_t}$ for every s_t in $\hat{G}$, which is corresponding to the tth row of $\hat{\boldsymbol{R}}_s$, preserving the sequential relationship among every local structure in $\hat{G}$. Different from embedding the edge formation, we do not build a decoder here because local structures in $\hat{G}$ do not have strict pair-wise relationships and the encoder has already been able to learn their sequential relationships pretty well.

To encode the asynchronous evolution timing of every local structure into the structure embedding $\hat{\boldsymbol{R}}_s$, we train the above attention model by a regression task, estimating the evolution time interval $\delta_{s_t, s_{t'}}$ between any two local structures s_t and $s_{t'}$. To keep the article organization consistent, we present the details of the regression task and discuss the criteria of improving the training efficiency in Section 4.4.2.

4.3 Representation Fusion

To obtain the representation of vertex v_i in one of its occurring edge sequences s_t, we fuse v_i's edge formation embedding $\hat{\boldsymbol{r}}_{v_i}$ together with its structure embedding $\hat{\boldsymbol{r}}_{s_t}$ by summing them up and get $\acute{\boldsymbol{r}}_{v_i} = \hat{\boldsymbol{r}}_{v_i} + \hat{\boldsymbol{r}}_{s_t}$. Then, we input it into a FFN with the $ReLU(\cdot)$ activation to encode the final vertex representation as shown in Eq. (31).

$$\boldsymbol{r}_{\boldsymbol{v_i}} = LN(\text{ReLU}(\acute{\boldsymbol{r}}_{v_i}^T \boldsymbol{W}_f^7 + \boldsymbol{b}_{v_i}^7 + \acute{\boldsymbol{r}}_{v_i}^T)\boldsymbol{W}_f^8 + \boldsymbol{b}_{v_i}^8) \tag{31}$$

$\boldsymbol{W}_f^7, \boldsymbol{W}_f^8 \in \mathbb{R}^{k \times k}$ and $\boldsymbol{b}_{v_i}^7, \boldsymbol{b}_{v_i}^8 \in \mathbb{R}^{1 \times k}$ are the trainable weights and bias of this FFN. Since each vertex v_i will likely appear in multiple edge sequences, we will present how to train the TADGE by using all edge sequences containing v_i to update its representation in next subsection.

4.4 Training TADGE

In order to effectively embed the asynchronous structural evolution, we train the TADGE model by three self-supervised tasks jointly.

4.4.1 Training Task 1: Self-identification for Edge Formation Embedding

We employ a vertex classification task to train the Time-aware Transformer for learning the edge formation embedding. Intuitively, no matter how v_i's connections change, its embedding $\hat{\boldsymbol{r}}_{v_i}$ should well identify v_i itself since it is representing v_i's edge formation information instead of other vertices. We employ a softmax with cross-entropy loss to train the Time-aware Transformer well-classifying vertices' own identity as shown in Eq. (32).

$$\begin{aligned}\mathcal{L}_v = -\sum_{v_i \in s_t} \sum_{s_t \in \hat{G}} \Big(&\boldsymbol{y}_{v_i}^T \log\left(\text{softmax}\left(\hat{\boldsymbol{R}}_v \hat{\boldsymbol{r}}_{v_i}\right)\right) \\ &+ \left(1 - \boldsymbol{y}_{v_i}^T\right) \log\left(1 - \text{softmax}\left(\hat{\boldsymbol{R}}_v \hat{\boldsymbol{r}}_{v_i}\right)\right)\Big)\end{aligned} \tag{32}$$

$\hat{\boldsymbol{R}}_v$ contains the edge formation embeddings of all vertices in s_t. $\boldsymbol{y}_{v_i} \in \mathbb{R}^{|V| \times 1}$ is the self-identification of v_i in one-hot encoding that the ith element in $\boldsymbol{y}_{v_i}$ is 1 and others are 0.

4.4.2 Training Task 2: Evolution Time Interval Regression for Structural Evolution Embedding

Embedding the evolution starting time is the key to preserve the asynchronous structural evolution. We employ a regression task on approximating the evolution time interval $\delta_{s_t, s_{t'}}$ between any pair of local structures s_t and $s_{t'}$ while learning their structure embeddings $\hat{\boldsymbol{r}}_{s_t}$ and $\hat{\boldsymbol{r}}_{s_{t'}}$ in $\hat{\boldsymbol{R}}_s$. Mathematically, this regression task is written in Eq. (33), where $\boldsymbol{w}_s \in \mathbb{R}^{k \times 1}$ is the trainable linear projection weights.

$$\mathcal{L}_s = \frac{1}{m(m-1)} \sum_{s_t \in \hat{G}} \sum_{\substack{s_{t'} \in \hat{G} \\ t' < t}} \left(\boldsymbol{w}_s^T \left(\hat{\boldsymbol{r}}_{s_t} + \hat{\boldsymbol{r}}_{s_{t'}}\right) - \delta_{s_t, s_{t'}}\right)^2 \tag{33}$$

The benefits of regressing the evolution time interval are twofold. First, it well preserves the sequential dependency among local structures while merely embedding the absolute evolution starting time cannot achieve. Second, it augments the scale of training samples so that the model is easy to converge and avoids under-fitting. However, there are $m(m-1)/2$ pairs of local structures being used to compute the gradient of Eq. (33). When m becomes extremely large, the gradient calculation will take a long time, which is an efficiency bottleneck of training the TADGE.

Inspired by the negative sampling, we adopt a sliding window with length ℓ_s to sample the structure sequence $\hat{G} = \{s_1, s_2, \cdots, s_m\}$ and construct structure pairs within the siding window as training samples. The sliding step size $\tilde{o}$ should satisfy $1 \leq \tilde{o} < \ell_s$ for ensuring two consecutive subsequences having overlap. Otherwise, the continuous evolution of the whole dynamic graph will be intermittent. Thus, the maximum number of sliding windows m_w satisfies $m_w \leq m - \ell_s + 1$. Since there are $\ell_s(\ell_s - 1)/2$ training samples in each sliding window, $m_w \ell_s(\ell_s - 1)/2$ training samples are obtained in total. According to the Lemma 1 and 2, when the sliding windows satisfy $\ell_s^2 - 3\ell_s + 3 < m$, the scale of training samples will definitely be reduced

while ensuring the overlap of sliding windows, therefore improving the training efficiency.

Lemma 1 $m_w \frac{\ell_s(\ell_s-1)}{2} < \frac{m(m-1)}{2}$ *when* $m_w < \frac{(m-1)^2}{(\ell_s-1)^2}$.

Proof: *Since* $m > \ell_s > 1$, *we have* $\frac{m}{\ell_s} < \frac{m-1}{\ell_s-1}$. *From* $m_w \frac{\ell_s(\ell_s-1)}{2} < \frac{m(m-1)}{2}$, *we get* $m_w < \frac{m(m-1)}{\ell_s(\ell_s-1)} < \frac{(m-1)^2}{(\ell_s-1)^2}$.

Lemma 2 *When* $m > \ell_s > 1$, $m_w \leq m - \ell_s + 1$ *and* $m_w \frac{\ell_s(\ell_s-1)}{2} < \frac{m(m-1)}{2}$ *simultaneously establish if* $\ell_s^2 - 3\ell_s + 3 < m$.

Proof: *Since* $m - \ell_s + 1$ *is the upper bound of* m_w *and* $m_w \frac{\ell_s(\ell_s-1)}{2} < \frac{m(m-1)}{2}$ *established when* $m_w < \frac{(m-1)^2}{(\ell_s-1)^2}$, *we let* $m - \ell_s + 1 < \frac{(m-1)^2}{(\ell_s-1)^2}$ *and get* $m - \ell_s < \frac{(m-1)^2}{(\ell_s-1)^2} - 1 = \frac{(m-\ell_s)(m+\ell_s-2)}{(\ell_s-1)^2}$. *Therefore,* $(\ell_s-1)^2 < m+\ell_s-2$, *and, finally, we have* $\ell_s^2 - 3\ell_s + 3 < m$.

4.4.3 Training Task 3: Time-aware Edge Reconstruction for Final Representations

Edge reconstruction task has been widely adopted to train the static graph embedding algorithms. It assumes that the representation of connected vertices should be close. However, this assumption oversimplified the edge formation process in the dynamic graph since the temporal information such as ToV and ToE has been neglected. A vertex can join the dynamic graph many times forming edges with the same pair of vertices but having different ToV and ToE. In the dynamic graph, when vertices are connected and have similar ToV and ToE, their representation should be close.

We propose a new task, namely time-aware edge reconstruction, to simultaneously estimate the ToE while reconstructing the edges between pairs of vertices. Given the final representation $\boldsymbol{R}_v = [\boldsymbol{r}_{v_1}, \boldsymbol{r}_{v_2}, \cdots]^T \in \mathbb{R}^{|V|\times k}$, the time-aware edge reconstruction not only classifies whether there is an edge between any pair of vertices, but also regresses the corresponding ToE at the same time. The objective function is shown in Eq. (34), where $\mathcal{L}_{edg}$ and $\mathcal{L}_{ToE}$ are for the edge reconstruction and the ToE regression, respectively.

$$\mathcal{L}_r = \mathcal{L}_{edg} + \mathcal{L}_{ToE} \tag{34}$$

Similar to identifying the vertex itself in Section 4.4.1, we again build a softmax with cross-entropy loss for edge reconstruction as shown in Eq. (35), where $y_{v_i,v_j} = 1$ if there is an edge between v_i and v_j in s_t, otherwise $y_{v_i,v_j} = 0$.

$$\begin{aligned}\mathcal{L}_{edg} = -\sum_{v_i,v_j\in s_t}\sum_{s_t\in\hat{G}} \Big(& y_{v_i,v_j}\log\left(\text{softmax}\left(\boldsymbol{r}_{v_i}^T\boldsymbol{r}_{v_j}\right)\right) \\ &+ \left(1-y_{v_i,v_j}\right)\log\left(1-\text{softmax}\left(\boldsymbol{r}_{v_i}^T\boldsymbol{r}_{v_j}\right)\right)\Big)\end{aligned} \tag{35}$$

$$\mathcal{L}_{ToE} = \frac{1}{2\hat{m}}\sum_{v_i,v_j\in s_t}\sum_{s_t\in\hat{G}}\left(\boldsymbol{w}_{ToE}^T\left(\boldsymbol{r}_{v_i}+\boldsymbol{r}_{v_j}\right)-\delta_{v_i,v_j}\right)^2 \tag{36}$$

$\mathcal{L}_{ToE}$ in Eq. (36) is a ToE regression that approximates the ToE δ_{v_i,v_j} of the edge linking v_i and v_j by using their representations $\boldsymbol{r}_{v_i}$ and $\boldsymbol{r}_{v_j}$. $\boldsymbol{w}_{ToE} \in \mathbb{R}^{k\times 1}$ is the trainable linear projection weights, and $\hat{m}$ is the number of training edges. When we minimize $\mathcal{L}_r$ in Eq. (34), the optimal results will be obtained if and only if both the edge reconstruction loss in Eq. (35) and the ToE regression loss in Eq. (36) reach the minimum, eventually making vertices' representation close if they are connected and have similar ToV and ToE.

4.4.4 Optimization Strategy

As TADGE is built upon the deep neural network structure, we initialize the representation R_v by using DeepWalk [17] and then apply Adam, a mini-batch stochastic gradient descent optimizer, to learn the model parameters by minimizing the joint loss:

$$\mathcal{L} = \mathcal{L}_v + \mathcal{L}_s + \mathcal{L}_r \tag{37}$$

The DeepWalk is trained by a static graph constructed from the edges in the training set. Thanks to the above three training tasks, the asynchronous structural evolution will be gradually embedded into the vertex representation. Meanwhile, we normalize the ToE and the time interval between structures to $[0, 1)$ by an arc-cotangent function $\delta = 2\arctan(\delta)/\pi$, suppressing the effects of a very large value of δ on the model convergence.

It is worth mentioning that adding weighting factors to each sub-objective in Eq. (37) can balance the scale difference of their effects on the overall loss and result in better performance. However, the weighting factors are application-specific. Therefore, in this work, we assign equal weights to each sub-objective, training a general model regardless of the specific applications. In addition, negative sampling can be applied to the sub-objectives $\mathcal{L}_v$ in Eq. (32) and $\mathcal{L}_{edg}$ in Eq. (35) to improve the training efficiency.

4.4.5 Training Protocol

To properly train the TADGE, we feed all edge sequences $\{s_1, s_2, \cdots, s_t, \cdots, s_m\}$ into the TADGE one by one to update its trainable parameters. Every time when an edge sequence s_t is fed into the TADGE, we update the representation of vertices in the s_t while keeping that of vertices not in the s_t unchanged. As a result, when the TADGE is trained on all edge sequences, each vertex v_i's representation will be updated by every edge sequence where v_i is in.

When updating the representation of a vertex v_i in an edge sequence s_t, we must ensure that v_i can merely see those vertices that joined the s_t before v_i. Therefore, we introduce the attention mask $\boldsymbol{M}$ in Eq. (12) and (18) when learning the edge formation embedding. This requirement must also be met when learning the structure embedding for v_i. To achieve this, we set $\hat{\boldsymbol{r}}_{v_j} = \hat{\boldsymbol{r}}_{v_{EOS}}$ for every $v_j \in s_t$ if $i < j \leq \ell$ when computing $\boldsymbol{h}_{s_t}$ in Eq. (24) before learning the structure embedding for v_i, thus ensuring only existed local structures are visible to v_i.

5 Experiments

In this section, we validate the effectiveness of our proposed TADGE in three public real-world datasets and benchmark against the state-of-the-art baseline methods on several data mining applications.

5.1 Experimental Setting

5.1.1 Datasets

We benchmark the TADGE algorithm in three public real-world datasets, whose properties are introduced below.

- **Transaction**[1] [18]. This is a dynamic bitcoin transaction network on the Bitcoin OTC platform. A vertex

1. https://snap.stanford.edu/data/soc-sign-bitcoin-otc.html

is a trader who buys and sells bitcoins. Two traders form an edge when they complete a transaction. The ToE is the time interval between buying and selling. Each trader is associated with a trustworthy label in low, middle, and high for classification.

- **Hyperlink**[2] [19]. This is a dynamic subreddit-to-subreddit hyperlink network extracted from the posts that create hyperlinks from one subreddit to another on Reddit. The vertex is a subreddit and the edge is a hyperlink connecting two subreddits. Each vertex has a binary semantic label for classification.
- **Discussion**[3] [20]. It is a dynamic discussion network extracted from a stack exchange website. The vertex is a user who posts, replies, comments and answers questions on the website. Once a user interact with others, an edge is formed between them.

The statistics of these datasets are presented in Table 1.

TABLE 1
Statistics of datasets

Dataset	$\|\mathbf{V}\|$	$\|\mathbf{E}\|$	Mean Degree	Mean ToE	Std. ToE	#Classes
Transaction	5,881	35,592	3.665	30.693 days	72.848	3
Hyperlink	54,075	571,927	7.701	58.350 days	121.937	2
Discussion	194,085	1,443,339	3.987	76.575 days	218.925	-

5.1.2 Baseline Methods

We first choose the popular random-walk-based graph embedding methods and graph neural network models as the common baselines. Both of them can only embed synchronous structural evolutions. Besides, we replace the Time-aware Transformer and the structure embedding in our TADGE with the graph attention network (GAT) [5] to compare the effectiveness of our TADGE against GAT while embedding the asynchronous structural evolutions. Lastly, we further evaluate the vertex representation merely learned by the Time-aware Transformer for exploring how TADGE benefits from the structure embedding.

- **DeepWalk**[4] [17]. DeepWalk is a classical static graph embedding approach based on the random-walk and a skip-gram algorithm.
- **CTDNE** [8]. The Continuous-Time Dynamic Network Embedding consists of a temporal random walk and a skip-gram algorithm to embed the continuous-time dynamic graph. It merely embeds the synchronous sequential edge formation but neglects the evolution timing and duration.
- **GraphSAGE**[5] [21]. This is a graph neural network approach for embedding the dynamic graph by computing the graph convolution from the vertices' connection change over time. We test different aggregators including GCN, mean, mean-pooling, and LSTM and report the best results in each dataset.
- **GAT**[6] [5]. This is one of the most popular attention-based approach for graph embedding, expanding the perception range of vertices from their local neighborhoods to all vertices in the whole graph. In order to benchmark our TADGE against GAT, We train the model with a loss function $\mathcal{L} = \mathcal{L}_v + \mathcal{L}_r$ so that the dynamic edge formation and the ToE are preserved by the GAT for comparison.
- **GAT-strc** We supplement the structure embedding presented in Section 4.2 into the above GAT, thus making it preserve the asynchronous structural evolutions. We train the model by using the same loss function $\mathcal{L}$ as the TADGE for a fair comparison.
- **TADGE-tTran** We learn the vertex representation by the Time-aware Transformer which merely preserves the dynamic edge formation with the ToE. We remove the structural attention from the TADGE while setting $\hat{\boldsymbol{r}}_{s_t} = 0$ and $\mathcal{L}_s = 0$ to train the model.

We do not benchmark the TADGE against those methods regarding the SGS as inputs. Our graph model contains finer-grained edge information than the snapshot graphs. Due to the time granularity issues of the SGS, we fail to construct the same training and test sets as TADGE for a fair comparison.

5.1.3 Experiment Setup

To split training and test sets, we first randomly select a set of vertices appearing at m different timestamps following the uniform distribution. Next, we randomly select 20% of them as the starting vertices to sample the dynamic graph G by using the time-centrality-biased temporal random walk presented in Section 3. The obtained edge sequences will be treated as the test set. After that, we remove the test edges from the dynamic graph and sample it again starting from the rest 80% of vertices to construct a training set, eventually ensuring that no test edges occur in the training set and all edge sequences are different in both training and test sets. Due to different scales of datasets, we configure the temporal random walk with the settings in Table 2.

TABLE 2
Parameter Setting of Temporal Random Walk

Dataset	m	Max. ℓ	Min. ℓ
Transaction	10,000	5	3
Hyperlink	200,000	5	3
Discussion	300,000	10	3

We set the minimum walk length to 3 so that there are at least 2 edges in every edge sequence indicating the structural evolutions. Besides, we found that vertices have limited multi-hop connections in the dynamic graph, which is different from that of in a static graph. Over 98% of multi-hop connections in all three datasets do not exceed the maximum walk length. We construct static graphs from the edges respectively in the training and test sets to train and test the DeepWalk and GraphSAGE. Since the first step in CTDNE and GAT is sampling the dynamic graph as edge sequences, we skip this step and directly train them with the edge sequences in our obtained training set. Hence, TADGE and all baselines are trained by the same set of edges or edge sequences for a fair comparison.

2. https://snap.stanford.edu/data/soc-RedditHyperlinks.html
3. https://snap.stanford.edu/data/sx-superuser.html
4. https://github.com/thunlp/OpenNE
5. https://github.com/williamleif/GraphSAGE
6. https://github.com/PetarV-/GAT

All experiments are conducted with $k = 128$ as the dimension of representation vectors for the TADGE and all baseline methods in three datasets. Following the recommended parameter setting of the Transformer in [12], we set the number of head $\bar{o} = 8$ and stack $N = 6$ blocks for the Time-aware Transformer and GAT to learn the embeddings in the Hyperlink and Discussion datasets. Due to the small scale of the Transaction data, we set $\bar{o} = 4$ and $N = 3$. In training, each batch contains 200, 500, and 300 edge sequences for Transaction, Hyperlink, and Discussion, respectively. The learning rate of the Adam gradient descent optimizer is set to 0.005. We adopt the optimal parameters for DeepWalk, CTDNE and GraphSAGE, which presented in the original papers. Both TADGE and baseline methods are trained with enough epochs for ensuring the convergence. All experiments are conducted on a standard workstation with Intel Xeon Gold 5122 CPUs, an RTX2080TI GPU, and 32GB RAM.

5.1.4 Evaluation Metrics

Evaluating Classification Performance. We adopt Micro-F1 and Macro-F1 scores as evaluation metrics for the classification tasks. Mathematically, they are defined as below:

$$\text{MicroF1} = \frac{2\sum_{i=1}^{c} \text{TP}_i}{\sum_{i=1}^{c} (2\text{TP}_i + \text{FP}_i + \text{FN}_i)} \tag{38}$$

$$\text{MacroF1} = \frac{1}{c}\sum_{i=1}^{c} \frac{2\text{TP}_i}{2\text{TP}_i + \text{FP}_i + \text{FN}} \tag{39}$$

where c is the total number of classes and TP_i, FP_i, and FN_i respectively are the true positive, false positive, and false negative of the predicted results for the ith class. In multi-class classification, the Micro-F1 scores measure the overall classification accuracy regardless of the performance in classifying individual classes. The Macro-F1 scores are the mean of class-wise F1 scores in which it is sensitive to the classification performance of minority classes while the Micro-F1 scores are not.

Evaluating Regression Performance. We evaluate the regression performance by measuring the Root Mean Square Error (RMSE) between the predicted values and the ground truths. Mathematically, they are defined as below.

$$\text{RMSE} = \sqrt{\frac{\sum_{y \in \mathcal{S}_{test}} (y - \hat{y})^2}{|\mathcal{S}_{test}|}} \tag{40}$$

RMSE reveals the regression error between the truth value y and the predicted one $\hat{y}$ in the test set $\mathcal{S}_{test}$.

5.2 Experimental Results and Analysis

We conduct experiments to validate the effectiveness and efficiency of our proposed TADGE and report the results.

5.2.1 ToE Prediction

Given the embedding of vertex pairs, the ToE prediction is estimating the ToE of edge they formed, thus testing how effectively the learned vertex representations preserve the temporal dynamics. For DeepWalk, CTDNE and GraphSAGE, which do not leverage ToE regression as a training task, we make use of the embeddings obtained in their training phases to train a regression model that employs $\mathcal{L}_{ToE}$ in Eq. (36) as the objective function and adds an elastic net as the regularization.

TABLE 3
RMSE of ToE prediction

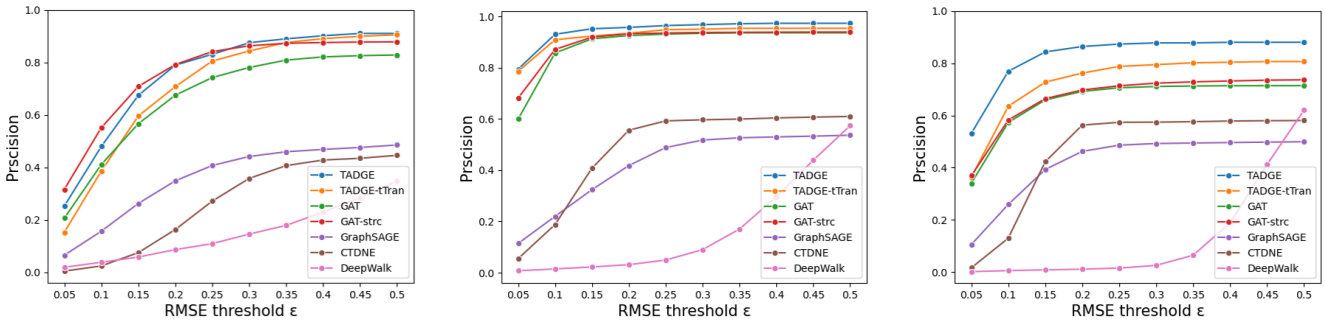
	Transaction	Hyperlink	Discussion
DeepWalk	0.5586	0.4667	0.4602
CTDNE	0.4397	0.4582	0.4718
GraphSAGE	0.4465	0.4464	0.4748
GAT	0.4304	0.4125	0.3394
GAT-strc	0.4025	0.4099	0.3317
TADGE-tTran	0.3959	0.4073	0.3298
TADGE	**0.3795**	**0.4004**	**0.3170**

The test results of ToE prediction are presented in the Table 3. The lower the RMSE, the more accurate the ToE prediction. Our TADGE achieves the lowest RMSE and dramatically outperforms all baseline methods in three datasets, demonstrating that the temporal dynamic has been well preserved by the TADGE. Comparing to those merely embedding the dynamic edge formation, i.e., GAT and TADGE-tTran, the ToE prediction error drops a lot when they further embed the asynchronous evolutions of local structures, thus validating the benefits of preserving asynchronous structural evolutions in ToE prediction.

5.2.2 Static Edge Prediction

The objective of static edge prediction is to determine whether a pair of vertices will form an edge in future timestamps when giving their current representations. This task ignores the ToV, which is widely adopted by existing work to validate the performance of embedding algorithms in preserving the linkage structures of the graph. We employ the cosine distance to measure the similarity of vertices' representation and predict whether they form an edge by a sigmoid function.

There are two ways to conduct the experiments of edge prediction. One is to treat it as a binary classification problem, predicting whether an edge exists with the learned representations of two vertices. Precision and recall are usually employed as evaluation metrics. This protocol focuses more on evaluating how many edges the model can correctly predict while neglecting each vertex's prediction performance. Therefore, even when the model fails in predicting edges for the minority of vertices, it will not affect the overall precision and recall much. Another experiment protocol treats the edge prediction as a multi-class classification problem, identifying whether any given vertex will form an edge with all the rest vertices. Micro-F1 and Macro-F1 scores are usually adopted to measure the prediction performance. The Micro-F1 score measures the overall prediction accuracy regardless of the individual performance of each vertex, which achieves the same goal as the former experiment protocol. Meanwhile, the Macro-F1 score indicates the vertex-wise average performance regardless of how often they appear in the graph. Therefore, this experiment protocol can reveal the performance of models in edge prediction much more comprehensively than the former one. In this study, we adopt the latter one for edge prediction and the results are summarized in Table 4.



(a) Results in the Transaction dataset (b) Results in the Hyperlink dataset (c) Results in the Discussion dataset

Fig. 4. Precision of the time-aware edge prediction with varying RMSE threshold ε.

TABLE 4
Results of Static Edge Prediction

	Transaction		Hyperlink		Discussion	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
DeepWalk	0.8783	0.7012	0.8587	0.6476	0.8486	0.5115
CTDNE	0.5177	0.4103	0.7253	0.4527	0.6897	0.3839
GraphSAGE	0.6132	0.5157	0.7114	0.3732	0.6125	0.3788
GAT	0.8300	0.8033	0.9362	0.9071	0.7140	0.6794
GAT-strc	0.8787	0.8627	0.9389	0.9140	0.7377	0.7050
TADGE-tTran	0.9111	0.8864	0.9536	0.9328	0.8060	0.7721
TADGE	**0.9133**	**0.8939**	**0.9732**	**0.9612**	**0.8799**	**0.8567**

TABLE 5
Results of Vertex Classification

	Transaction		Hyperlink	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1
DeepWalk	0.5889	0.3721	0.7179	0.5185
CTDNE	0.5969	0.3680	0.7119	0.4653
GraphSAGE	0.6037	0.3711	0.7939	0.7361
GAT	0.5952	0.4256	0.7156	0.6705
GAT-strc	0.6173	0.4424	0.8067	0.7562
TADGE-tTran	0.6042	0.4122	0.7596	0.7247
TADGE	**0.6302**	**0.4644**	**0.8140**	**0.7634**

Our TADGE achieved the highest Micro-F1 and Macro-F1 scores in all three datasets. In our Time-aware Transformer, it first learns vertices' self-attentions within the local structure by an encoder. Regarding this encoded vertex attentions as the context information, a decoder is then built to further embed the exact connections between the vertex pairs within the local structure, thus better preserving the dynamic edge formation and resulting in higher F1 scores than other baseline methods. Besides, embedding the asynchronous structural evolutions further boosts the effectiveness of preserving the dynamic edge formation. Consequently, our TADGE dramatically outperforms all baseline methods, especially in the Discussion dataset which is with the largest scale and highly dynamic connections.

5.2.3 *Time-aware Edge Prediction*

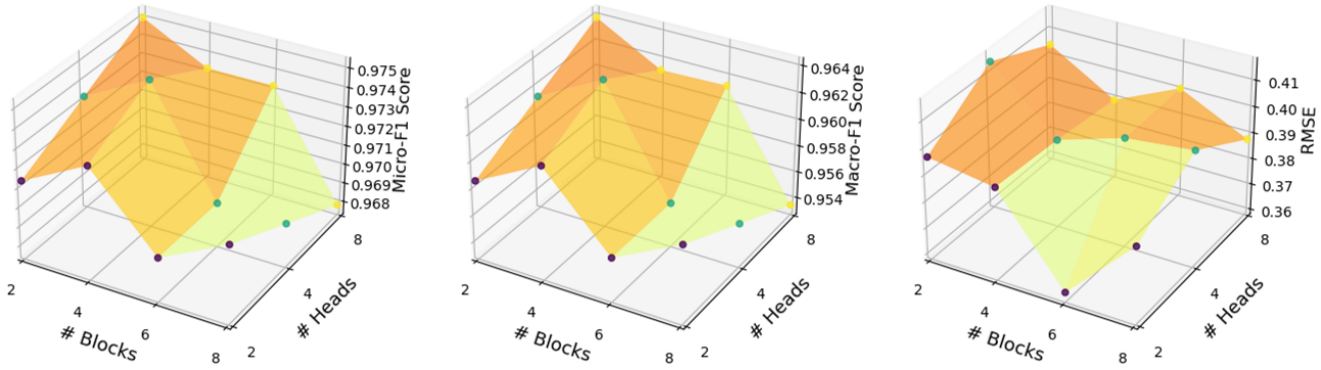
Time-aware edge prediction is a specific application for dynamic graph embedding abstracted from many data mining applications such as forecasting future crowd flow and recommending items at varying time intervals. It simultaneously performs static edge prediction and ToE prediction. An edge is correctly predicted if and only if true positive results are obtained in static edge prediction and the RMSE of ToE estimation is smaller than a threshold ε. In order to expose the effect of ToE prediction error on the performance of the time-aware edge prediction, we test the threshold ε from 0.05 to 0.5. Because the performance of ToE prediction will not affect the results of negative edge samples in the time-aware edge prediction, we conduct the experiment on the positive edge samples in the test set and report the overall precision in Fig. 4.

Our TADGE performs the best in the Hyperlink and the Discussion when $\varepsilon \geq 0.05$. This shows that the RMSE of ToE prediction for most edges predicted by our TADGE in these datasets is less than 0.05. Although the TADGE performs slightly worse than the GAT-strc in the Transaction with small ε, it becomes the best of all when $\varepsilon > 0.25$. The results of DeepWalk show totally different trend due to its poor performance in ToE prediction. The performance superiority of TADGE against DeepWalk, CTDNE and GraphSAGE demonstrates that temporal information carried by the ToV and ToE is a key aspect of preserving the dynamics connection between vertices. Consequently, our TADGE preserves the spatial and temporal dynamics of the edge formation very well, therefore resulting in superior performance in the time-aware edge prediction.

5.2.4 *Vertex Classification*

Vertex classification aims to identify the unique labels of the vertices using their learned embeddings. A support vector machine (SVM) with a Gaussian kernel is trained by using the embeddings obtained from the training set with the corresponding vertex labels. After obtaining vertices' embeddings in the test set, we input them into the well-trained SVM and classify their labels. It tests how well the embedding algorithm is in preserving the evolutionary patterns. Since the Discussion dataset does not contain vertex labels, we compare the classification performance measured by the Micro-F1 and Macro-F1 scores in both Transaction and Hyperlink datasets.

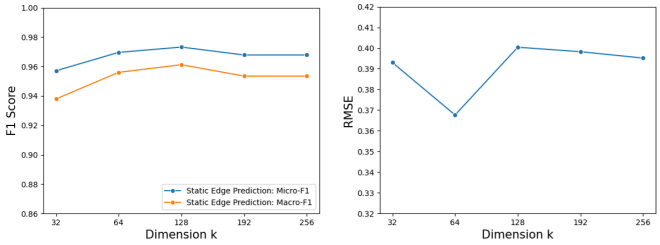
The results are presented in Table 5. At the first glance, it is clear that our TADGE achieves the highest Micro-F1 and Macro-F1 scores in both datasets. Remarkably, when neglecting the asynchronous structural evolving and merely embedding the pair-wise dynamic connection changes, the classification accuracy drops significantly. Since TADGE and GAT-strc are trained by the same loss functions, the performance improvement of TADGE comes from the excellent



(a) Micro-F1 scores of static edge prediction (b) Macro-F1 scores of static edge prediction (c) RMSE of ToE prediction

Fig. 5. Testing the hyper-parameters with varying blocks N and heads $\bar{o}$ in the Hyperlink.

preservation of dynamic connection changes by the Time-aware Transformer and super expression ability of our proposed attention mechanism. Comparing to CTDNE and GraphSAGE, the superior performance of TADGE demonstrates that embedding the temporal information, i.e., ToV and ToE, makes the embeddings more discriminative. In summary, our TADGE is capable of well preserving the asynchronous structural evolution patterns of vertices, thus leading to better vertex classification performance.



(a) Results in the self-identification of vertex and static edge prediction (b) Results in the ToE prediction

Fig. 6. Testing the dimension of representation k in the Hyperlink.

5.2.5 Parameter Sensitivity Analysis

We investigate the performance fluctuations of TADGE with varied hyper-parameters. In particular, we study the sensitivity of TADGE to the embedding dimension k and the number of blocks N and heads $\bar{o}$ in static edge prediction and ToE prediction using the Hyperlink dataset. We vary the value of one hyper-parameter while fixing the others.

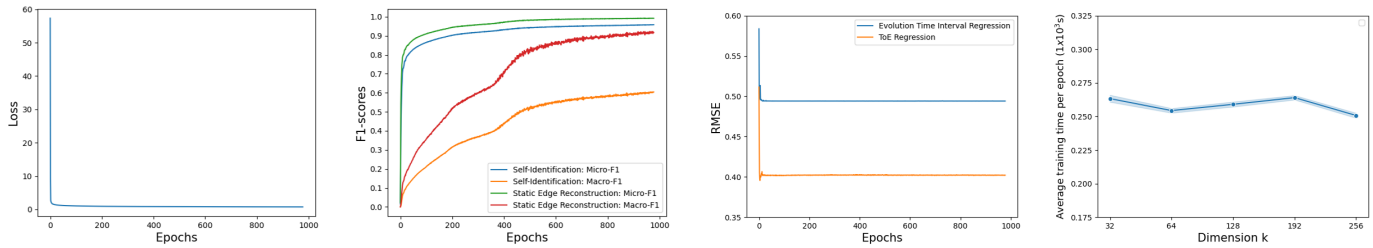
Impact of the number of blocks N and heads $\bar{o}$. The number of blocks and heads are the key parameters affecting the expression ability of the self-attention mechanism in the TADGE to learn the asynchronous structural evolutions. We test the combination of $N \in \{2, 4, 6, 8\}$ and $\bar{o} \in \{2, 4, 8\}$ to evaluate their impacts on the performance fluctuations of TADGE. As the results in Fig. 5, in the static edge prediction, the increasing number of blocks has a negative impact on both Micro-F1 and Macro-F1 scores while, in general, more heads the better performance. Similar trends are observed in ToE prediction. In summary, TADGE prefers more heads but fewer blocks so as to better embed the asynchronous structural evolutions in the dynamic graph.

Impact of k. The embedding dimension k is an important hyper-parameter affecting the expressiveness of TADGE. We exterminate k in $\{32, 64, 128, 192, 256\}$. As shown in Fig. 6, in static edge prediction, both Micro-F1 and Macro-F1 scores increase at the beginning, indicating that the expressive power of the embedding grows as k increases After reaching the best F1 score when $k = 128$, the classification performance slightly drops afterward. In ToE prediction, TADGE gets the smallest RMSE when $k = 64$, and the prediction errors with the other k values are close. Hence, TADGE is not sensitive to k in ToE prediction.

5.2.6 Convergence and Training Efficiency Analysis

We demonstrate the convergence and training efficiency of TADGE in the Hyperlink datasets. In Fig. 7(a), we observe that training loss drops quickly and converges within a hundred epochs. The overall training accuracy in both edge reconstruction and self-identification of vertices quickly converges to over 0.9 Micro-f1 score as shown in Fig. 7(b). Although the loss changes slightly, the Macro-f1 scores in both tasks gradually increase and eventually reach the convergence, which is consistent with the conclusion drawn in [12]. Because the appearing times of vertices in a dynamic graph follow the long-tailed distribution, there are a number of vertices having very few connections to others. They fail to provide enough linkage information for the embedding algorithms to learn. Consequently, in the self-identification tasks, the TADGE converges to very high Micro-F1 scores but with relatively low Macro-F1 scores. Better dealing with this minority of vertices will be a direction for further extending this study. In both ToE prediction and structural evolution time interval estimation, which are two regression tasks, the training RMSE converges within 20 epochs as shown in Fig. 7(c), indicating the advantageous convergence speed of the TADGE in the regression tasks.

Fig. 7(d) shows the average running time of every epoch while training the TADGE with varying dimension k. The shadow shows the variance of training time. As k increases, the running time of updating the trainable parameters of TADGE at each epoch fluctuates slightly. This validates that the training time is not sensitive to the dimension of the representations. Hence, we conclude that TADGE has very



(a) Training loss (b) Training F1 scores (c) Training RMSE (d) Training time with varying k

Fig. 7. Convergence and training efficiency of TADGE in the Hyperlink. The shadowed area in (d) shows the variance of training time.

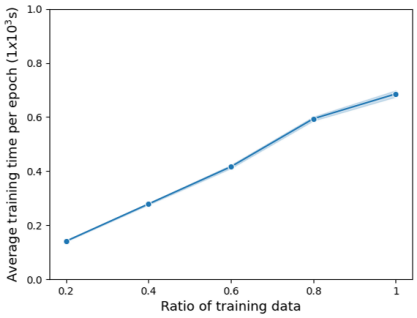


Fig. 8. Training time of TADGE per epoch with varying data proportions.

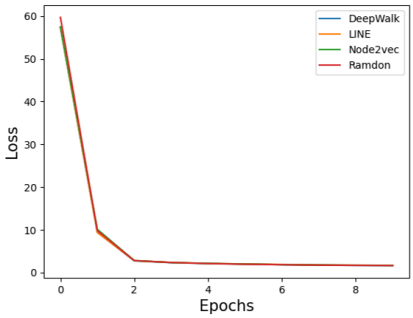


Fig. 9. Training loss of TADGE using different initialization methods.

good convergence and training efficiency.

5.2.7 Scalability Analysis

We conduct a scalability test for the TADGE in the Discussion dataset which contains $194,085$ vertices and $1,443,339$ edges. We vary the proportion of the training data in $\{0.2, 0.4, 0.6, 0.8, 1.0\}$ to train the TADGE and report the average training time of every epoch. Ideally, the training time should increase linearly when we enlarge the scale of training data. The growth of training time is shown in Fig. 8. The line indicates the average running time of updating the weights of TADGE at each epoch and the shadow shows the variance. As the scale of training data gradually enlarges, the average training time per epoch grows linearly from 0.141×10^3 seconds to 0.686×10^3 seconds. Consequently, we conclude that the TADGE has very good scalability for embedding very large-scale dynamic graphs.

5.2.8 Analysis of Initialization Methods for Training TADGE

To evaluate the impact of initialization approaches on the effectiveness and training efficiency of TADGE, we respectively employ DeepWalk [17], Node2vec [22], and LINE [23] to initialize the representation R_v and train TADGE in the Hyperlink dataset. All these approaches are trained on a static graph constructed from the edges in the training set. We also adopt a random initialization following the normal distribution as one of the baselines. The training loss is shown in Fig. 9. Although the initial loss of using static graph embedding approaches as the initialization is smaller than that of using a random one, the training loss shows almost the same trend with a very slight difference since the first epoch. This confirms that the outstanding convergence speed of TADGE is because of the approach itself rather than the initialization approach. Table 6 reports

TABLE 6
Results of TADGE with Varying Initialization Methods in the Hyperlink

	Static Edge Prediction		Vertex Classification		ToE Prediction
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	RMSE
Random	0.9649	0.9511	0.8276	0.7550	0.4029
DeepWalk	0.9732	0.9612	0.8140	0.7634	0.4004
Node2vec	0.9666	0.9534	0.8123	0.7598	0.3984
LINE	0.9631	0.9485	0.8246	0.7703	0.4007

the results of TADGE in the ToE prediction, static edge prediction, and vertex classification when initializing the training using different approaches. The results are close to each other, demonstrating that our TADGE is not sensitive to the initialization approaches.

6 Related Work

The main issue in dynamic graph embedding is capturing and encoding the dynamic evolving nature of vertices and edges. Modeling the dynamic graph in a proper manner is the foundation as it captures the dynamics of vertices and edges for later embedding. Existing approaches model the dynamic graph as either a snapshot graph sequence (SGS)) [1] [2] [3] [4] [5] [6] [7] or neighborhood formation sequences (NFS) sampled from the temporal random walk [8] [9] [10] [11]. These approaches merely capture the synchronous structural evolutions and their limitations have been discussed in Section 1. Our dynamic graph model captures not only the dynamic connection changes among vertices but also the asynchronous evolution starting time and duration for embedding.

Given the above dynamic graph models, the embedding algorithm aims to encode the captured evolution patterns as representations of vertices or edges. TNE [1] is a pioneer work that embeds the structural difference in SGS. A series

of similar approaches are continuously published, such as DHPE [3], TMF [24], and DynGraph2Vec [25]. Instead of measuring the overall difference between snapshots, DynamicTriad [4] embeds the triad formation process among vertices when a graph evolves. GraphSAGE [21] learns the structure by a graph neural network, leveraging neighborhood information to generate representations for the new coming vertices. Du, et al. [26] proposed an generic framework to extend skip-gram-based static graph embedding methods to update vertices' representations when the graph evolves and generate embeddings for new vertices. PME [27] and MGCN [28] borrowed the similar ideas of embedding SGS, learning the correlation among multiple sub-graphs for highly accurate link prediction.

Learning the sequential edge formation was originally proposed in [8] which embedding continuous-time dynamic graph. HTNE [11] samples NFS by a multivariate Hawkes process and employ an attention network to embed sequential edge formation. Following this idea, a series works have been proposed to learn the sequential patterns from edge sequences [9] [10] by using the graph attention [5] [29] [9], generative adversarial networks [7] [30]. TMER [31] proposed temporal meta-path to obtain sequential patterns for recommendation. Although none of them embed the asynchronous structural evolutions, they inspire us to design the TADGE for jointly embedding the pair-wise connections and the local structures.

There are a few works focusing on temporal network embedding. M^2DNE [32] embeds the temporal edge formation process and the evolving scale of the graph. It introduces a time decay function while calculating the attention value between connected vertices, which treats the temporal information as an edge attribute for embedding. EPNE [33] embeds the periodic connection changes among vertices by causal convolutions. TCDGE [34] co-trains a liner regressor to embed ToE while learning the connection difference between consecutive snapshots. Although references [35] and [36] claim themselves as the temporal network embedding, they degenerate temporal networks as NFS before embedding, thus merely preserving sequential structural evolutions without temporal information. None of above mentioned works deals with the dynamic ToV and ToE at the same time. Hence, they fail to embed the asynchronous structural evolutions but we fill this research gap in this paper.

7 Conclusions

We generically model a dynamic graph as a set of temporal edges, appending the respective joining time of vertices (ToV) and timespan of edges (ToE). A time-centrality-biased temporal random walk is proposed to sample the dynamic graph as a set of temporal edge sequences for capturing the asynchronous structural evolutions. A TADGE model containing a Time-aware Transformer and a structural embedding model is then proposed to simultaneously embed the dynamic connection changes with ToE and the asynchronous evolution starting time of every local structure. The experimental results show that our TADGE achieves significant performance improvement over the state-of-the-art approaches in various data mining tasks, thus validating the effectiveness of TADGE to embed the asynchronous structural evolutions. Besides, TADGE is very efficient and scalable when handling large-scale dynamic graphs.

Acknowledgment

The work is supported by Shenzhen-Hong Kong-Macau Technology Research Programme Type C (No. SGDX20201103095203029), Hong Kong Research Grants Council under Theme-based Research Scheme (No. T41-603/20-R), Research Impact Fund (No. R5034-18), Collaborative Research Fund (No. C5026-18G), and General Research Fund (No. 16202722), Natural Science Foundation of China (No. 62072125), PolyU Research and Innovation Office (No. BD4A), the Australian Research Council under ARC Future Fellowship (No. FT210100624), Discovery Project (No. DP190101985), Discovery Early Career Research Award (No. DE200101465, DE230101033), Industrial Transformation Training Centre (No. IC200100022), and Centre of Excellence (No. CE200100025), and was partially conducted in Research Institute for Artificial Intelligence of Things at PolyU and the JC STEM Lab of Data Science Foundations funded by The Hong Kong Jockey Club Charities Trust.

References

[1] L. Zhu, D. Guo, J. Yin, G. Ver Steeg, and A. Galstyan, "Scalable temporal latent space inference for link prediction in dynamic social networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 10, pp. 2765–2777, 2016.

[2] P. Goyal, S. R. Chhetri, and A. Canedo, "dyngraph2vec: Capturing network dynamics using dynamic graph representation learning," *Knowledge-Based Systems*, vol. 187, p. 104816, 2020.

[3] D. Zhu, P. Cui, Z. Zhang, J. Pei, and W. Zhu, "High-order proximity preserved embedding for dynamic networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 11, pp. 2134–2144, 2018.

[4] L. Zhou, Y. Yang, X. Ren, F. Wu, and Y. Zhuang, "Dynamic network embedding by modeling triadic closure process," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[5] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations*, 2018.

[6] F. Manessi, A. Rozza, and M. Manzo, "Dynamic graph convolutional networks," *Pattern Recognition*, vol. 97, p. 107000, 2020.

[7] Y. Xiong, Y. Zhang, H. Fu, W. Wang, Y. Zhu, and S. Y. Philip, "Dyngraphgan: Dynamic graph embedding via generative adversarial networks," in *International Conference on Database Systems for Advanced Applications*, 2019, pp. 536–552.

[8] G. H. Nguyen, J. B. Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim, "Continuous-time dynamic network embeddings," in *Companion Proceedings of the The Web Conference 2018*, 2018, pp. 969–976.

[9] A. Fathy and K. Li, "Temporalgat: Attention-based dynamic graph representation learning," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2020, pp. 413–423.

[10] X. Chang, X. Liu, J. Wen, S. Li, Y. Fang, L. Song, and Y. Qi, "Continuous-time dynamic graph learning via neural interaction processes," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 145–154.

[11] Y. Zuo, G. Liu, H. Lin, J. Guo, X. Hu, and J. Wu, "Embedding temporal network via neighborhood formation," in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, 2018, pp. 2857–2866.

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

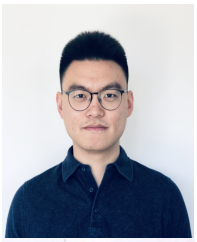
[13] H. Chen, H. Yin, T. Chen, Q. V. H. Nguyen, W.-C. Peng, and X. Li, "Exploiting centrality information with graph convolutions for network representation learning," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, 2019, pp. 590–601.

[14] Y. Gu, Y. Sun, Y. Li, and Y. Yang, "Rare: Social rank regulated large-scale network embedding," in *Proceedings of the 2018 World Wide Web Conference*, 2018, pp. 359–368.
[15] Z. Wang, Y. Ma, Z. Liu, and J. Tang, "R-transformer: Recurrent neural network enhanced transformer," *arXiv preprint arXiv:1907.05572*, 2019.
[16] I. M. Baytas, C. Xiao, X. Zhang, F. Wang, A. K. Jain, and J. Zhou, "Patient subtyping via time-aware lstm networks," in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017, pp. 65–74.
[17] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 701–710.
[18] S. Kumar, F. Spezzano, V. Subrahmanian, and C. Faloutsos, "Edge weight prediction in weighted signed networks," in *Data Mining (ICDM), 2016 IEEE 16th International Conference on*, 2016, pp. 221–230.
[19] S. Kumar, W. L. Hamilton, J. Leskovec, and D. Jurafsky, "Community interaction and conflict on the web," in *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, 2018, pp. 933–943.
[20] A. Paranjape, A. R. Benson, and J. Leskovec, "Motifs in temporal networks," in *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, 2017, pp. 601–610.
[21] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, 2017, pp. 1024–1034.
[22] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.
[23] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th international conference on world wide web*, 2015, pp. 1067–1077.
[24] W. Yu, C. C. Aggarwal, and W. Wang, "Temporally factorized network modeling for evolutionary network analysis," in *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, 2017, pp. 455–464.
[25] P. Goyal, S. R. Chhetri, and A. Canedo, "dyngraph2vec: Capturing network dynamics using dynamic graph representation learning," *Knowledge-Based Systems*, vol. 187, p. 104816, 2020.
[26] L. Du, Y. Wang, G. Song, Z. Lu, and J. Wang, "Dynamic network embedding: An extended approach for skip-gram based network embedding." in *IJCAI*, vol. 2018, 2018, pp. 2086–2092.
[27] H. Chen, H. Yin, W. Wang, H. Wang, Q. V. H. Nguyen, and X. Li, "Pme: projected metric embedding on heterogeneous networks for link prediction," in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, 2018, pp. 1177–1186.
[28] H. Chen, H. Yin, X. Sun, T. Chen, B. Gabrys, and K. Musial, "Multi-level graph convolutional networks for cross-platform anchor link prediction," in *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, 2020, pp. 1503–1511.
[29] A. Sankar, Y. Wu, L. Gou, W. Zhang, and H. Yang, "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*, 2020, pp. 519–527.
[30] D. Zhou, L. Zheng, J. Han, and J. He, "A data-driven graph generative model for temporal interaction networks," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 401–411.
[31] H. Chen, Y. Li, X. Sun, G. Xu, and H. Yin, "Temporal meta-path guided explainable recommendation," in *Proceedings of the 14th ACM international conference on web search and data mining*, 2021, pp. 1056–1064.
[32] Y. Lu, X. Wang, C. Shi, P. S. Yu, and Y. Ye, "Temporal network embedding with micro-and macro-dynamics," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019, pp. 469–478.
[33] J. Wang, Y. Jin, G. Song, and X. Ma, "Epne: Evolutionary pattern preserving network embedding," in *Proceedings of the 24th European Conference on Artificial Intelligence*, 2020, pp. 1603–1610.
[34] Y. Yang, J. Cao, M. Stojmenovic, S. Wang, Y. Cheng, C. Lum, and Z. Li, "Time-capturing dynamic graph embedding for temporal linkage evolution," *IEEE Transactions on Knowledge and Data Engineering*, 2021.
[35] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein, "Temporal graph networks for deep learning on dynamic graphs," *arXiv preprint arXiv:2006.10637*, 2020.
[36] S. Kumar, X. Zhang, and J. Leskovec, "Predicting dynamic embedding trajectory in temporal interaction networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 1269–1278.

Yu Yang is currently a Research Assistant Professor with the Department of Computing, The Hong Kong Polytechnic University. He received the Ph.D. degree in Computer Science from The Hong Kong Polytechnic University in 2021. His research interests include spatiotemporal data analysis, representation learning, urban computing, and learning analytics.

Hongzhi Yin received the Ph.D. degree in computer science from Peking University in 2014. He is an Associate Professor with the University of Queensland. He received the Australia Research Council Discovery Early-Career Researcher Award, in 2015. His research interests include recommendation system, user profiling, topic models, deep learning, social media mining, and location-based services.

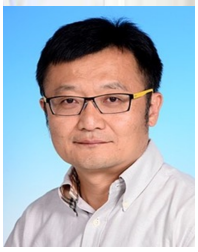
Jiannong Cao (M'93-SM'05-F'15) received the B.Sc. degree in Computer Science from Nanjing University, China, in 1982, and the M.Sc. and Ph.D. degrees in Computer Science from Washington State University, USA, in 1986 and 1990, respectively. He is the Chair Professor with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong. His current research interests include big data analytics, edge intelligence, and mobile computing.



Tong Chen received the Ph.D. degree in computer science from the University of Queensland in 2020. He is a lecturer with Data Science Group, School of ITEE, UQ. His research work has been published on top venues like SIGKDD, ICDE, WWW, IJCAI, AAAI, TKDE, etc., where his research interests include data mining, recommender systems, and predictive analytics.

Nguyen Quoc Viet Hung is a Senior Lecturer in Griffith University. He earned his Master and Ph.D. degrees from EPFL (Switzerland). He received the Australia Research Council Discovery Early-Career Researcher Award, in 2020. His research focuses on Data Integration, Information Retrieval, Trust Management, Recommender Systems, etc., with special emphasis on web data, social data and sensor data.

Xiaofang Zhou is a Chair Professor with the Department of Computer Science and Engineering at Hong Kong University of Science and Technology. He received the Ph.D. degree in Computer Science from University of Queensland in 1994. His research focus is to find effective and efficient solutions for analysing large-scale data for business, scientific and personal applications.



Lei Chen received the Ph.D. degree in Computer Science from the University of Waterloo, Canada, in 2005. He is a Chair Professor with the Department of Computer Science and Engineering at Hong Kong University of Science and Technology. His research interests include data-driven machine learning, graph, and crowdsourcing.