TP-GNN: Continuous Dynamic Graph Neural Network for Graph Classification

Jie Liu
College of Computer and Information Science
College of Software
Southwest University
Chongqing, China
lj2000828@email.swu.edu.cn

Jiamou Liu
School of Computer Science
University of Auckland
Auckland, New Zealand
jiamou.liu@auckland.ac.nz

Kaiqi Zhao
School of Computer Science
University of Auckland
Auckland, New Zealand
kaiqi.zhao@auckland.ac.nz

Yanni Tang
School of Computer Science
University of Auckland
Auckland, New Zealand
ytan370@aucklanduni.ac.nz

Wu Chen*
College of Software
Southwest University
Chongqing, China
chenwu@swu.edu.cn

2024 IEEE 40th International Conference on Data Engineering (ICDE) | 979-8-3503-1715-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICDE60146.2024.00215

***Abstract*—Dynamic networks are data structures that represent the interactions among various entities in real-world systems, with their topology and node properties evolving over time. However, prevailing approaches typically derive node embeddings through aggregating temporal neighbor nodes of adjacent several hops, thus failing to capture the long temporal dependencies in dynamic networks. Furthermore, existing research on dynamic networks focuses on node- and edge-level tasks, lacking the support of graph-level tasks. To address the limitations of current approaches, this paper proposes TP-GNN, a novel continuous dynamic graph neural network model intended for graph classification in dynamic networks, which offers two primary advantages: (1) TP-GNN captures the long temporal dependencies via a novel message-passing method based on the information flow among the nodes, and (2) it learns the network evolution process from edge order for accurate dynamic network analytics. We evaluate the performance of TP-GNN in five datasets, including a new dataset we created from a Java software project. The results show that our method outperforms state-of-the-art approaches in graph classification with an average improvement of 4.91% in terms of F_1 Score [1].**

***Index Terms*—dynamic graph neural networks, dynamic networks, dynamic graphs, graph classification**

I. Introduction

A **dynamic network** is a graph whose nodes and edges can be added, removed, and changed over time [10]. Compared to static networks, dynamic networks incorporate additional temporal information [28], making them suitable tools for modeling dynamic interactions between system components. Numerous studies have leveraged dynamic networks for link prediction, node classification, edge classification, and other graph analysis tasks, thus benefitting real-world applications such as sequential recommendations [27], [34], [48], anomaly detection [16], [29], and community discovery [24].

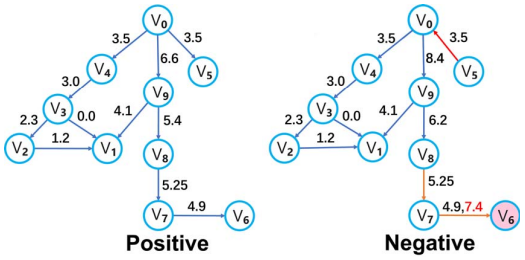

Fig. 1: The figure depicts two session networks from the Forum-java dataset, where the left network is positive and the right is negative. Yet, if ignoring the timestamps on the edges, the two networks are identical.

Towards precise dynamic graph analytics, learning an effective dynamic network representation is a pivotal task. As such, many dynamic graph neural networks (DGNNs) have been proposed. Earlier DGNNs, such as DySAT [27] and EvolveGCN [22] model the time dimension in a discrete manner. They treat a dynamic network as a series of static snapshots and use Recurrent Neural Networks (RNNs) [1] with Graph Neural Networks (GNNs) to extract temporal and structural information, respectively. Hence, this inevitably leads to a loss of edge order and temporal clusters (i.e., frequent co-occurrence of links) in snapshots [28]. Recently, continuous DGNNs have been proposed to analyze dynamic networks in a continuous way. TGAT [39] introduces a functional time coding technique rooted in Bochner's theorem and harnesses self-attention mechanisms to model dynamic networks. TGN [25] combines memory module and graph convolution operation to propose a general and efficient dynamic network representation learning framework. DyGNN [18] designs update and propagation components to capture the network evolution. Besides, GraphMixer [7] utilizes Multi-Layer Perceptron (MLP) as the main framework and only aggregates the "most recent 1-hop neighbor" to simplify the model architecture for link prediction in dynamic networks.

*Corresponding author

[1]Codes and dataset are available at *https://github.com/Jie-0828/TP-GNN*.

2375-026X/24/$31.00 ©2024 IEEE
DOI 10.1109/ICDE60146.2024.00215

Although the aforementioned DGNNs have achieved promising results in modeling dynamic networks, they have three limitations.

1) **Lack of support for graph-level tasks**. Graph-level analysis is commonly needed in real-world dynamic networks. For instance, in log anomaly detection, according to previous research [33], [42], events generated by a user request can be constructed as a dynamic *session network*, where nodes represent log events and edges encapsulate the association between events (See Fig. 1). Likewise, each user's places of interests (POIs) can be formulated to a dynamic user-trajectory network, with nodes representing the user's check-in locations and edges tracing the users' movements from one POI to another. Analysis conducted solely at the node- and edge-levels in these dynamic networks often falls short in effectively discerning abnormal system/user behavior [12], which requires the analysis of the dynamic network as a whole. Thus anomaly detection can be regarded as a dynamic graph classification problem. However, existing DGNNS are only designed for node or edge-level tasks. To the best of our knowledge, no existing solution is available for graph-level tasks in dynamic networks.
2) **Limited receptive field for capturing long temporal dependencies**. Edges in real-world dynamic networks often exhibit strong temporal correlations [8], [19]. Existing approaches only capture local information by aggregating *"h-hop"* neighbor nodes, where *h* is usually a small value. Therefore, such methods lose long temporal dependencies in the learned representations. Fig. 1 shows a normal and an abnormal log session graph in a software system. Traditional message-passing methods obtain the embeddings of v_6 in both graphs by aggregating the information of its first-order temporal neighbor v_7 [23], [25], [39]. Since the two graphs have identical topology, existing message-passing methods cannot distinguish well the two graphs. From the perspective of information flow, v_6 receives the v_7 information twice (i.e., at $t = 4.9$ and 7.4, respectively). Initially, at $t = 4.9$, v_7 lacks information from v_8 and v_9 due to their later occurrences. However, at the second time, v_7 should include information from v_8 and v_9 generated at time steps prior to $t = 7.4$. Intuitively, the difference between the two graphs occurs after the log event of v_9. Thus, the information from v_9, v_8 is essential to obtain distinct representations for v_6 and v_7 in the two graphs. The lack of consideration of long temporal dependencies residing in the information flow limits the ability of current message-passing methods to learn distinguishable network representations.
3) **Overlooking global edge sequencing in modeling network evolution**. Existing DGNNs either employ RNNs on graph snapshots or modify the message-passing mechanism to capture temporal information [17]. However, these methods tend to learn node embeddings by aggregating the temporal neighbors, thereby capturing only local temporal information (for nodes or edges), overlooking edge sequencing in the evolution process of the dynamic networks. The order of timestamped edges implies the global network evolution process that helps to distinguish different dynamic networks. For example, in Fig. 1, the edge sequence of the normal graph is $V_3 \rightarrow V_1, V_2 \rightarrow V_1, ..., V_7 \rightarrow V_6, V_8 \rightarrow V_7, V_9 \rightarrow V_8$, $V_0 \rightarrow V_9$. Compared with the normal graph, the abnormal graph has one more edge $V_7 \rightarrow V_6$ after $V_9 \rightarrow V_8$.

To enable dynamic graph classification tasks, we propose a new continuous dynamic graph neural network, called *Temporal Propagation-Graph Neural Network* (TP-GNN) in Sec. IV-A. To address the second limitation, TP-GNN employs a novel message-passing method, namely *Temporal Propagation*. Given a dynamic network, temporal propagation follows the direction of information flow in capturing the long temporal dependencies within the dynamic networks (See Sec. IV-B). This method helps the learned representations of the nodes encode distinct temporal features for different graph classes. To address the third limitation, we propose a *Global Temporal Embedding Extractor* to learn network evolution from edge sequences. Unlike existing DGNNs that employ RNNs to capture temporal dependencies among graph snapshots, our approach first converts the node representations obtained from temporal propagation into edge representations and then feeds them into the Gated Recurrent Unit (GRU) according to the chronological order of edge establishment (See Sec. IV-C). In this way, TP-GNN captures the network evolution process, thus improving the accuracy of dynamic graph classification. We summarize the contributions of this paper as follows:

- We propose an end-to-end continuous DGNN for graph classification in dynamic networks. To the best of our knowledge, this is the first work to design a continuous dynamic graph neural network for the graph-level task.
- We design a new message-passing technique for dynamic networks. This method respects the direction of information flow in the network for message-passing, capturing long temporal dependencies.
- We develop a Global Temporal Embedding Extractor to extract and incorporate the sequential information of edges. This approach informs our model about the network evolution, leading to higher classification accuracy.
- We conduct experiments on five datasets, which include a log dataset created by ourselves from an open-access Java software project. Experimental results show the superiority of our model over state-of-the-art methods in dynamic graph classification, yielding a maximum increase of 9.86% in terms of F_1 Score.

II. Related Work

In recent years, due to the high efficiency of Graph Neural Networks (GNNs) in capturing node correlations through neighborhood-based aggregation strategies [37], various models have been proposed for different application scenarios, such as GCN [14], GAT [32], and GraphSage [11]. These

well-established message-passing mechanisms are designed to eliminate ordering among edges, thus enabling the acquisition of order-invariant node representations. However, this characteristic renders them unsuitable for dynamic networks, where the establishment order of edges carries vital information. To cope with the evolution of the graph over time, endeavors have been made in dynamic graph representation learning, including matrix factorization-based [9], random walk-based [21], [35], and GNN-based (DGNN) [41]. Among these, DGNN possesses robust generalization and expressive capabilities, so has garnered widespread favor among researchers.

According to the time granularity of modeling dynamic networks, DGNN can be divided into discrete DGNN and continuous DGNN [28]. Discrete DGNN generally involves cropping the dynamic network into a series of independent, static snapshots based on a given time window. Such model then combines various forms of deep time-series modeling with GNNs to learn the network representation [17]. For instance, DySAT [27] and Addgraph [45] utilized GCN and RNN to extract structural and temporal information, respectively. EvolveGCN [22] used RNN to evolve GCN parameters to capture the dynamics of snapshot sequences. TADDY [16] used a transformer-based framework to learn edge representations from network snapshots with coupled spatial-temporal patterns. Recently, GC-LSTM [3] harnessed the adjacency matrix of each snapshot as an input to LSTM for effectively tracking structural changes.

In contrast, continuous DGNN directly models dynamic networks formed by continuous-timestamped edges, avoiding the information loss caused by partitioning snapshots [41]. This approach thus is more flexible in terms of modeling potential. DyREP [30] utilized a temporal point process to capture the event association and communication within a dynamic network. TGAT [39] integrated self-attention mechanisms and functional time coding based on Bochner's theorem to handle complex temporal patterns. DyGNN [18] used the update and propagation components based on LSTM to propagate information about dynamic node interactions. TREND [36] built a new dynamic graph representation learning framework based on the Hawkes process to capture the event influence. Furthermore, TGN [25] proposed an efficient continuous DGNN framework for simulating dynamic network evolution, while TIGER [44] introduced dual storage modules on the basis of TGN to better exploit neighborhood information and alleviate the staleness problem. Recently, GraphMixer [7] proposed an MLP-based DGNN framework that effectively filtered out spurious or noisy information by aggregating the "most recent 1-hop neighbor".

Comparision: Our proposed model belongs to continuous DGNNs but distinguishes itself in three key aspects from the aforementioned methodologies: 1) TP-GNN introduces a novel message-passing mechanism aligned with the direction of information flow. This approach enables the model to capture long temporal dependencies within the network, thus generating distinct node embeddings for different types of graphs. 2) In contrast to previous approaches using GRU to capture the temporal dependence between snapshots, we present a GRU-based method to capture network evolution from edge sequences. 3) Our method addresses the graph classification problem within dynamic graphs for the first time, setting it apart from existing works that focus on node- or edge-level analysis of dynamic graphs.

III. Problem Formulation

The continuous-time dynamic network is a directed graph whose topology and node properties change over time, which implies that one or more edges are established between two nodes at different timestamps; and possibly even at the same timestamp. In such a network, a node $v \in \mathcal{V}$ signifies a system entity in the real world, such as a log event or a user's POI. A $\mathcal{T}$-labeled temporal edge (u, v, t) indicates that a connection from u to v is established at time t. Notice that the direction of the edge denotes the information flow. In the log session network, nodes $u, v \in \mathcal{V}$ represent log events, and an edge from u to v denotes that event v occurs after event u. This sequence potentially delineates a relationship between log events, such as a failure of a network request causing an application to crash. Similarly, for a user-trajectory network, a node in $\mathcal{V}$ denotes the POI and an edge from u to v represents a user moving from u to v, also denoting a form of information flow. Hence, continuous-time dynamic network is formally defined as:

Definition 1 (Continuous-Time Dynamic Network)**.** *A* **Continuous-Time Dynamic Network (CTDN)** *is a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}^{\mathcal{T}}, \mathcal{X}, \mathcal{T})$, where $\mathcal{V}$ is a set of nodes, $\mathcal{T} \in \mathbb{R}$ is a set of timestamps, $\mathcal{X}$ is a $n \times q$ feature matrix that associates a q-dimensional feature vector $\mathcal{X}(v)$ with each node $v \in \mathcal{V}$. $\mathcal{E}^{\mathcal{T}} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of $\mathcal{T}$-labeled edges, and each edge $e^t_{uv} = (u, v, t) \in \mathcal{E}^{\mathcal{T}}$ is the interaction between nodes u and v at time t.*

The research task of this paper is to solve the graph classification problem in dynamic networks. With the notation above, we formulate this objective as a binary classification problem as follows:

Definition 2 (Graph Embedding Function)**.** *Let $\mathbb{G}$ denote the set of all possible dynamic networks. A* **graph embedding function** *is $f\colon \mathbb{G} \to \mathbb{R}^d$, which maps a dynamic network $\mathcal{G}$ to a d-dimensional vector $\boldsymbol{g} = f(\mathcal{G})$.*

Definition 3 (Dynamic Graph Classification)**.** *Given N dynamic networks $\{\mathcal{G}_i\}_{i=1}^N$, each belongs to a class $Y_{\mathcal{G}_i} \in \{0, 1\}$, i.e., the ground-truth graph label of $\mathcal{G}_i$.* **The dynamic graph classification** *task aims to learn a function $\varphi\colon \mathbb{R}^d \to \{0, 1\}$, which makes the predicted graph label $\varphi(\boldsymbol{g}_i)$ as close as possible to the true graph label $Y_{\mathcal{G}_i}$ for any $\mathcal{G}_i$.*

IV. Temporal Propagation - Graph Neural Network

A. Overall Architecture

The architecture of TP-GNN is portrayed in Fig. 2, which encompasses two main components: the temporal propagation

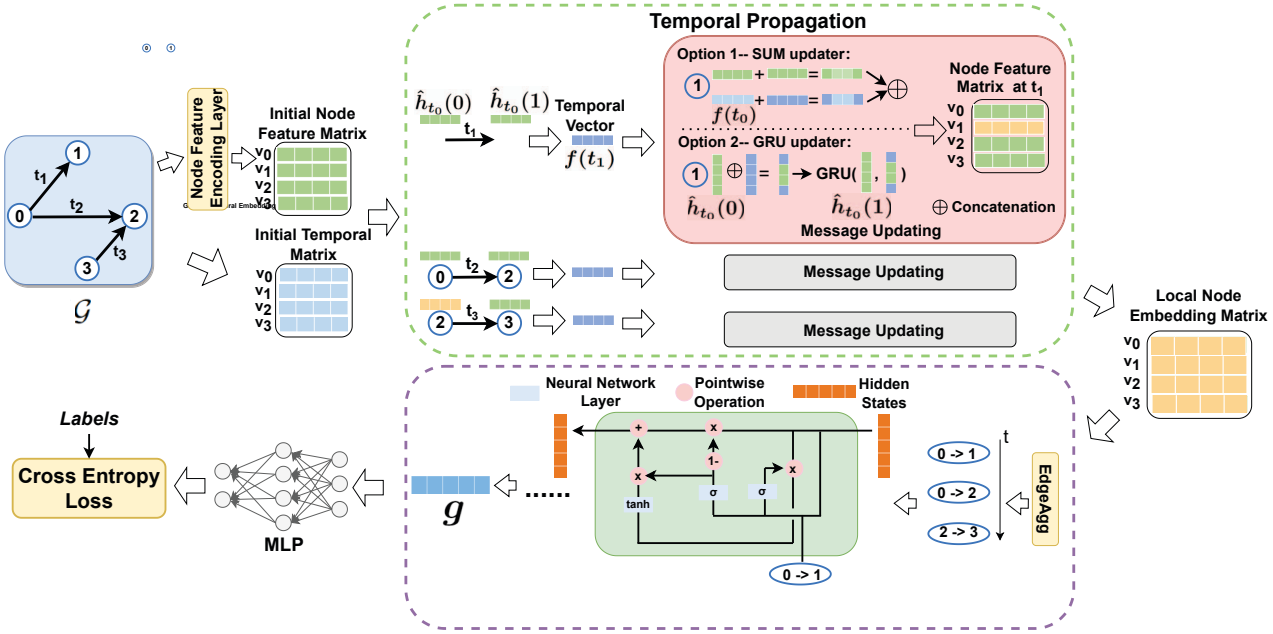

Fig. 2: The framework of TP-GNN consists of two main components: the temporal propagation (green dotted box) and the global temporal embedding extractor (purple dotted box). Within the temporal propagation component, we provide two alternative methods for node feature updating (red box). Here only shows the process of modeling a dynamic network.

(Sec. IV-B) and the global temporal embedding extractor (Sec. IV-C). This model inputs a set of dynamic networks $\{\mathcal{G}_i\}_{i=1}^{N}$ and finally outputs the predicted class label for each graph. For each network $\mathcal{G}_i \in \mathbb{G}$, the graph embedding $\boldsymbol{g}_i$ is typically learned through the following two modules:

1) The first module is a novel message-passing mechanism, called *temporal propagation*. The approach aggregates the features of neighbor nodes according to the direction of information flow in the network. In this way, the long temporal dependencies and structural information can be captured in the local node embedding matrix $\boldsymbol{H}$.
2) The second module is the *global temporal embedding extractor*. The extractor models the network evolution from the global edge sequence, where each edge is represented by the average of its source and target node embeddings in $\boldsymbol{H}$. The output is the graph embedding $\boldsymbol{g}_i$ for graph $\mathcal{G}_i$.

B. Temporal Propagation

In a dynamic network, the order of edge establishment often reflects the information flow in data. This order can furnish the model with important spatial-temporal information, helping to understand and analyze the dynamic behavior in the network. However, the message propagation mechanisms in most DGNNs (as in TGAT [39]), only aggregate neighbors that are several hops adjacent at the current time. Therefore, these methods tend to consider the local information and fail to capture the long dependencies of nodes in the network, as demonstrated in Fig. 1. To address the aforementioned issue, we propose a novel message-passing mechanism called temporal propagation that excels at capturing long temporal dependencies in dynamic networks.

The proposed temporal propagation is inspired by the information flow among nodes in dynamic networks. Intuitively, the target node v of a temporal edge (u, v, t) is influenced by the information from the source node u and all precedents of u before time t. Formally, we define the *influential nodes* to a target node v as below.

Definition 4 (Influential nodes)**.** *Given a CTDN* $\mathcal{G} = (\mathcal{V}, \mathcal{E}^{\mathcal{T}}, \mathcal{X}, \mathcal{T})$*, a valid path is a sequence of edges* $(u_1, u_2, t_1), (u_2, u_3, t_2), \ldots, (u_k, u_{k+1}, t_k)$*, where* $0 < t_1 \leq t_2 \leq \ldots \leq t_k$*. A node* u *is said to be an* **influential node** *to* v *if a valid path exists from* u *to* v*.*

Intuitively, if node u is influential to v, certain information flows from u to v over time. The message-passing mechanism should thus aggregate the information of u to v. Conversely, if u is not influential to v, the message-passing mechanism should not aggregate information from node u to v.

In the temporal propagation module, we propagate the information from all influential nodes to the target node according to chronological order. This module consists of two key components: the node feature and time encoding layer, and the temporal propagation layer.

1) Node feature and Time Encoding Layer: Firstly, to help the model better understand the relationship between the features, we use an embedding layer to transform discrete original node features into continuous dense vector representations. In this way, the node feature encoding layer can improve the performance and generalization ability of TP-GNN, which can be formally defined as follows:

$$\boldsymbol{X} \coloneqq \boldsymbol{W_i}\boldsymbol{\mathcal{X}} + \boldsymbol{b_i}, \tag{1}$$

where the $\boldsymbol{\mathcal{X}}$ is the original node feature matrix, $\boldsymbol{W_i}$ and $\boldsymbol{b_i}$ is a set of learnable parameters in embedding layer.

Secondly, the key temporal information in dynamic networks is usually revealed by the interaction time between nodes. By analyzing this information, dynamic characteristics such as interaction frequency and node order can be inferred, which is significant for further analysis of dynamic networks. Hence, we transform the interaction time into a continuous vector representation for flexible embedding into the model. In particular, for the edge (u, v, t), we utilize Time2vec [13] as a time-encoding function to map the timestamp t of information interaction between nodes into an d_t-dimensional space and generate a *time embedding vector* $\boldsymbol{f(t)} \in \mathbb{R}^{d_t}$, where d_t is the dimension of time embedding vector. The formal expression of the time encoding layer is as follows:

$$\boldsymbol{f(t)} \coloneqq (\boldsymbol{\omega_0} t + \boldsymbol{\varphi_0}) \oplus \sin(\boldsymbol{\omega} t + \boldsymbol{\varphi}), \tag{2}$$

where $\boldsymbol{\omega}$, $\boldsymbol{\omega_0}$, $\boldsymbol{\varphi}$ and $\boldsymbol{\varphi_0}$ are a set of learnable parameters in Time2vec, $\oplus$ denotes concatenation operation.

2) Temporal Propagation Layer: Unlike previous works that aggregate the fixed h-hop neighbors to update the feature of target node [25], [39], [44], our temporal propagation layer follows the direction of information flow in a dynamic network for message passing. In general, the advantages of temporal propagation can be summarized as follows: First, temporal propagation ensures the consistency of information in the network and avoids time confusion in the process of information propagation. Secondly, temporal propagation aggregates the features of all influential nodes according to the order and direction of information flow and thus captures the long temporal dependencies. Finally, each neighbor of the node is calculated only once in temporal propagation, which avoids the overhead of repeatedly aggregating the same neighbor node under each timestamp and improves the efficiency of the model. In particular, the process of temporal propagation layer can be described as follows:

Consider a CTDN $\mathcal{G} = (\mathcal{V}, \mathcal{E}^{\mathcal{T}}, \mathcal{X}, \mathcal{T})$, we first list the edge in chronological order, i.e., (u_1, v_1, t_1), (u_2, v_2, t_2), …, (u_m, v_m, t_m) where $0 < t_1 \leq t_2 \leq \ldots \leq t_m$. Our temporal propagation algorithm then goes over this list, iteratively performing message passing along each edge. Intending to develop a versatile approach applicable to a broader range of scenarios, we propose two methods to update the feature of the target node:

(i) Temporal Propagation-SUM: Firstly, we choose sum method to aggregate neighbor node features, which is a common method in GNNs [15]. As this method involves message passing exclusively between connected nodes, it can be viewed as a spatial-based graph convolutional approach [47]. Temporal propagation-SUM aggregates the features of influential neighbor nodes to target nodes according to the propagation path. This addition operation integrates the information of all influential nodes on the basis of retaining the original information, thus improving the ability of the model to build long temporal dependencies. Specifically, for any $v \in \mathcal{V}$, the SUM-updater computes a $(q + d_t)$-dimensional feature vector $\boldsymbol{h_v}$, which is formally defined as follows:

- At $t_0 = 0$, define the *temporal matrix* $\hat{\boldsymbol{M}}_{\boldsymbol{t_0}}$ as an all-zero $n \times d_t$ - dimensional matrix that associates a d_t-dimentional time embedding vector $\hat{\boldsymbol{M}}_{\boldsymbol{t_0}}(\boldsymbol{v})$ with each node $v \in \mathcal{V}$. This matrix is used to store the temporal information of node interactions in subsequent operations. The node feature matrix $\hat{\boldsymbol{X}}_{\boldsymbol{t_0}}$ at time t_0 can be initialized as $\boldsymbol{X}$.
- When an interaction (i.e., edge) is established between nodes, we update the target node feature. For the interaction (u, v, t_i), where $i > 0$, we first utilize the time-encoding function to encode the timestamp t_i and obtain the time embedding vector $\boldsymbol{f(t_i)}$. The node feature vector $\hat{\boldsymbol{X}}_{\boldsymbol{t_i}}(\boldsymbol{v})$ of node v at time t_i is then iteratively updated by aggregating the information from its corresponding source node u at time t_{i-1}. Simultaneously, we update the temporal matrix $\hat{\boldsymbol{M}}_{\boldsymbol{t_i}}$ at time t_i with the time embedding vector $\boldsymbol{f(t_i)}$. This process can be formally defined as follows:

$$\hat{\boldsymbol{X}}_{\boldsymbol{t_i}}(\boldsymbol{v}) \coloneqq \hat{\boldsymbol{X}}_{\boldsymbol{t_{i-1}}}(\boldsymbol{u}) + \hat{\boldsymbol{X}}_{\boldsymbol{t_{i-1}}}(\boldsymbol{v}), \tag{3}$$

$$\hat{\boldsymbol{M}}_{\boldsymbol{t_i}}(\boldsymbol{v}) \coloneqq \boldsymbol{f(t_i)} + \hat{\boldsymbol{M}}_{\boldsymbol{t_{i-1}}}(\boldsymbol{v}), \tag{4}$$

  where $\hat{\boldsymbol{X}}_{\boldsymbol{t_{i-1}}}(\boldsymbol{u}), \hat{\boldsymbol{X}}_{\boldsymbol{t_{i-1}}}(\boldsymbol{v})$ are the features of node u, v before time t_i.
- Upon completion of message passing in accordance with the order of interactions, $\hat{\boldsymbol{X}}$ and $\hat{\boldsymbol{M}}$ are concatenated to obtain an $n \times (q + d_t)$ dimensional node representation matrix $\hat{\boldsymbol{H}}$:

$$\hat{\boldsymbol{H}} \coloneqq \left(\hat{\boldsymbol{X}} \oplus \hat{\boldsymbol{M}}\right), \tag{5}$$

  where $\oplus$ denotes the concatenation operator on the second axis of $\hat{\boldsymbol{X}}$ and $\hat{\boldsymbol{M}}$.

(ii) Temporal Propagation-GRU: Since temporal propagation-SUM ignores complex feature changes, it may cause information mixing during aggregating neighbor features in the case of long interaction sequences. Therefore, we propose a second method of using GRU to capture the evolution between node features to update the representation of the target node. GRU's gating mechanism serves as a potent safeguard against gradient vanishing or exploding, thereby equipping our model with improved capacity for handling large dynamic graphs [6]. The fundamental difference from TGN [25] is that our approach updates the feature of the target node unidirectionally according to the information flow. Therefore, for graph classification tasks, this message passing method can improve the model's understanding of dynamic graph structure evolution and classification performance, especially in distinguishing the positive and negative samples caused by different interaction sequences. For example, the proposed message passing can identify the difference caused by interaction between nodes v_0 and v_5 in the positive and negative samples outlined in Fig. 1, while TGN cannot. Concretely, for any $v \in \mathcal{V}$, the temporal propagation-GRU computes a q-dimensional embedding vector $\boldsymbol{h_v}$ as follows:

- At $t_0 = 0$, the embedding vector $\hat{\boldsymbol{h}}(\boldsymbol{v})$ of every node $v \in \mathcal{V}$ is set to initial value $\hat{\boldsymbol{h}}_{\boldsymbol{t_0}}(\boldsymbol{v}) \coloneqq \boldsymbol{X}(v)$, where $\boldsymbol{X}(v)$ represents the initial features of node v.
- For an edge (u, v, t_i), the temporal propagation-GRU selectively retains or ignores information from influential nodes through the gating mechanism. In particular, we

concatenate the time embedding vector $\boldsymbol{f(t_i)}$ and the feature of the source node $\hat{\boldsymbol{h}}_{\boldsymbol{t_{i-1}}}(\boldsymbol{u})$ to represent the information to be propagated to the target node from the source node u at time t_i. The temporal propagation-GRU takes the embedding vector for the target node at t_{i-1} and the information to be propagated as input, and outputs the updated embedding vector of the target node v at time t_i:

$$\hat{\boldsymbol{h}}_{\boldsymbol{t_i}}(\boldsymbol{v}) \coloneqq GRU(\hat{\boldsymbol{h}}_{\boldsymbol{t_{i-1}}}(\boldsymbol{v}), [\hat{\boldsymbol{h}}_{\boldsymbol{t_{i-1}}}(\boldsymbol{u}) \oplus \boldsymbol{f(t_i)}]). \quad (6)$$

- After the message passing process according to the chronological order of interactions, we obtain the node feature vector $\hat{\boldsymbol{h}}(\boldsymbol{v})$ for each node $v \in \mathcal{V}$, and then stack them into a node feature matrix $\hat{\boldsymbol{H}} \in \mathbb{R}^{n \times q}$.

By employing either of the two updating methods, we derive a feature matrix $\hat{\boldsymbol{H}}$. Subsequently, we apply the $tanh$ activation function to $\hat{\boldsymbol{H}}$ for nonlinear mapping, resulting in a *local node embedding matrix* $\boldsymbol{H}$. This matrix encompasses *local node embedding vector* $\boldsymbol{h(v)}$ for each $v \in \mathcal{V}$. For further elaboration, please refer to Algorithm 1.

3) Discussion on Temporal Propagation: Through the proposed temporal propagation, the node embeddings will be updated by influential nodes in the network, while other nodes will not contribute to the node representation learning. Next, we proceed to provide a theoretical analysis of this property.

Formally, we define a node v to be *independent* from another node u in the temporal propagation algorithm if the local node embedding vector $\boldsymbol{h(v)}$ remains unchanged even when we change the input feature vector $\boldsymbol{X}(u)$ of node u, while keeping other nodes' feature vectors unchanged. The following theorem establishes that any node that is not influential to v is also independent of v, and vice versa.

Theorem 1. *For any nodes, $u, v \in V$, u is influential to v if and only if v is not independent of u in the temporal propagation algorithm.*

Proof. Let us assume that u is influential to v. Consequently, there exists a valid path from u to v, denoted as $u = u_1, u_2, \ldots, u_k = v$. To ensure efficient message passing, the temporal propagation algorithm processes the edges in a specific order. For any $i \in \{1, \ldots, k-2\}$, the edge from u_i to u_{i+1} is handled before the edge from u_{i+1} to u_{i+2}. With each message communicated along the timestamp, we refer to it as an "iteration". This implies that after the update of $\hat{\boldsymbol{h}}(\boldsymbol{u_i})$, an iteration wherein $\hat{\boldsymbol{h}}(\boldsymbol{u_{i+2}})$ will also be updated accordingly. As a result, the input feature vector $\boldsymbol{X(u)}$ converges and aggregates into the local node embedding of v.

On the contrary, we assume that in temporal propagation, v is not independent of u. In such a case, at some iteration i_1, $\hat{\boldsymbol{h}}(\boldsymbol{v})$ receives an update, and the feature information of u is aggregated into $\hat{\boldsymbol{h}}(\boldsymbol{v})$. This indicates the existence of an in-neighbor v_1 of v that is dependent on u, with an edge $(v_1, v, t_1) \in \mathcal{E}^{\mathcal{T}}$. If $v_1 = u$, then the sequence (v_1, v) forms a valid path. Otherwise $\hat{\boldsymbol{h}}(\boldsymbol{v_1})$ is updated at some iteration $i_2 < i_1$. Consequently, this signifies the existence of an in-neighbor v_2 of v_1 that is not independent from u, with an edge $(v_2, v_1, t_2) \in \mathcal{E}^{\mathcal{T}}$ where $t_2 \leq t_1$. Following the same reasoning, if $v_2 = u$, the sequence (v_2, v_1, v) forms a valid path.

Algorithm 1: Temporal Propagation

Input: a dynamic network $\mathcal{G} = (\mathcal{V}, \mathcal{E}^{\mathcal{T}}, \mathcal{X}, \mathcal{T})$ with edges $(u, v, t) \in \mathcal{E}^{\mathcal{T}}$, where the size of $\mathcal{V}$ is n, and t represents the construction time of the edge.

Sort the edges in ascending order by timestamp: $(u_1, v_1, t_1), \ldots, (u_m, v_m, t_m)$ with $0 < t_1 \leq \ldots \leq t_m$;
$\boldsymbol{X} \coloneqq \boldsymbol{W_i}\boldsymbol{\mathcal{X}} + \boldsymbol{b_i}$;
if *Update Component=SUM* **then**
  Initiate the temporal matrix $\hat{\boldsymbol{M}}_{\boldsymbol{t_0}} \coloneqq 0$;
  Initiate the node feature matrix $\hat{\boldsymbol{X}}_{\boldsymbol{t_0}} \coloneqq \boldsymbol{X}$;
  for $e^{t_1}_{u_1 v_1} = (u_1, v_1, t_1)$ **to** $e^{t_m}_{u_m v_m} = (u_m, v_m, t_m)$ **do**
    Take the edge $e^{t_i}_{uv} = (u, v, t_i)$;
    $\boldsymbol{f(t_i)} \coloneqq (\boldsymbol{\omega_0} t_i + \boldsymbol{\varphi_0}) \oplus \sin(\boldsymbol{\omega} t_i + \boldsymbol{\varphi})$;
    $\hat{\boldsymbol{X}}_{\boldsymbol{t_i}}(\boldsymbol{v}) \coloneqq \hat{\boldsymbol{X}}_{\boldsymbol{t_{i-1}}}(\boldsymbol{u}) + \hat{\boldsymbol{X}}_{\boldsymbol{t_{i-1}}}(\boldsymbol{v})$;
    $\hat{\boldsymbol{M}}_{\boldsymbol{t_i}}(\boldsymbol{v}) \coloneqq \boldsymbol{f(t_i)} + \hat{\boldsymbol{M}}_{\boldsymbol{t_{i-1}}}(\boldsymbol{v})$;
  $\hat{\boldsymbol{H}} \coloneqq \left(\hat{\boldsymbol{X}} \oplus \hat{\boldsymbol{M}}\right) \in \mathbb{R}^{n \times (q + d_t)}$;
else if *Update Component=GRU* **then**
  Initiate the feature vector $\hat{\boldsymbol{h}}_{\boldsymbol{t_0}}(\boldsymbol{v}) \coloneqq \boldsymbol{X(v)}$;
  for $e^{t_1}_{u_1 v_1} = (u_1, v_1, t_1)$ **to** $e^{t_m}_{u_m v_m} = (u_m, v_m, t_m)$ **do**
    Take the edge $e^{t_i}_{uv} = (u, v, t_i)$;
    $\boldsymbol{f(t_i)} \coloneqq (\boldsymbol{\omega_0} t_i + \boldsymbol{\varphi_0}) \oplus \sin(\boldsymbol{\omega} t_i + \boldsymbol{\varphi})$;
    $\hat{\boldsymbol{h}}_{\boldsymbol{t_i}}(\boldsymbol{v}) \coloneqq GRU(\hat{\boldsymbol{h}}_{\boldsymbol{t_{i-1}}}(\boldsymbol{v}), [\hat{\boldsymbol{h}}_{\boldsymbol{t_{i-1}}}(\boldsymbol{u}) \oplus \boldsymbol{f(t_i)}])$;
  Stack all $\hat{\boldsymbol{h}}_{\boldsymbol{v}}$ into a node feature matrix $\hat{\boldsymbol{H}} \in \mathbb{R}^{n \times q}$;
Compute $\boldsymbol{H} \coloneqq \tanh\left(\hat{\boldsymbol{H}}\right)$;
return the local node embedding matrix $\boldsymbol{H}$;

Otherwise, $\hat{\boldsymbol{h}}(\boldsymbol{v_2})$ is updated at some iteration $i_3 < i_2$. By continuing this argument, we obtain a sequence $v_1, v_2, v_3, \ldots$ that eventually terminates at $v_k = u$. At this point, we have successfully identified a valid path $u = v_k, v_{k-1}, \ldots, v_1, v$, illustrating that u is influential to v. □

C. Global Temporal Embedding Extractor

The temporal propagation module produces node representations that capture long temporal dependencies between nodes. To obtain an overall representation of the whole network for graph-level classification, these node representations need to be aggregated in an effective way. We have shown that the order of edges implies essential information about the network evolution process in Fig. 1. In this paper, we capture the network evolution from edge sequences based on GRU to generate graph embeddings. This choice is motivated by GRU's parsimonious parameterization, which facilitates faster training and effectively mitigates issues related to vanishing and exploding gradients [6]. It is noteworthy that GRU can be replaced by other sequential models according to the characteristics of a dataset. For instance, one can choose Transformer [31] for large dynamic graphs to capture longer dependencies.

In particular, we opt for the most widely-used $Average$ method among the six $EdgeAgg$ methods introduced in [23]: {Average, Hadamard, Weighted-L_1, Weighted-L_2, Activation, Concatenation}, to convert the local node embedding matrix $\boldsymbol{H}$ to the *local edge embedding matrix* $\boldsymbol{S_{loc}}$. For each edge $e^{t_i}_{uv} = (u, v, t_i)$, its *local edge embedding vector* can be expressed as $\boldsymbol{S_{loc}(u, v, t_i)}$ obtained by the average value of the embedded nodes u and v at both ends of the edge. Since

different node update strategies in temporal propagation will generate node embedding matrices with different dimensions, we uniformly use $m \times k$ to represent the size of $\boldsymbol{S_{loc}}$, where m is the number of edges in the dynamic network and k is the dimension of the edge embedding vector. Subsequently, the matrix $\boldsymbol{S_{loc}}$ is fed into GRU successively according to the order of the establishment of timestamped edges. The overall process can be represented as follows:

$$\boldsymbol{z}_{(u,v,t_i)} = \sigma[\boldsymbol{W_z}\boldsymbol{S_{loc}}(\boldsymbol{u},\boldsymbol{v},\boldsymbol{t_i}) + \boldsymbol{U_z}\boldsymbol{S}_{(u,v,t_{i-1})} + \boldsymbol{b_z}], \quad (7)$$

$$\boldsymbol{r}_{(u,v,t_i)} = \sigma[\boldsymbol{W_r}\boldsymbol{S_{loc}}(\boldsymbol{u},\boldsymbol{v},\boldsymbol{t_i}) + \boldsymbol{U_r}\boldsymbol{S}_{(u,v,t_{i-1})} + \boldsymbol{b_r}], \quad (8)$$

$$\hat{\boldsymbol{S}}_{(u,v,t_i)} = \tanh[\boldsymbol{W_s}\boldsymbol{S_{loc}}(\boldsymbol{u},\boldsymbol{v},\boldsymbol{t_i}) + \boldsymbol{r}_{(u,v,t_i)} \circ \boldsymbol{U_s}\boldsymbol{S}_{(u,v,t_{i-1})} + \boldsymbol{b_s}], \quad (9)$$

$$\boldsymbol{S}_{(u,v,t_i)} = \boldsymbol{z}_{(u,v,t_i)} \circ \boldsymbol{S}_{(u,v,t_{i-1})} + (1 - \boldsymbol{z}_{(u,v,t_i)}) \circ \hat{\boldsymbol{S}}_{(u,v,t_i)}, \quad (10)$$

where $\boldsymbol{U}$, $\boldsymbol{W}$ and $\boldsymbol{b}$ are the weights and bias parameters of GRU layer, $\circ$ represents the $hadamard$ multiplication operation between matrices.

We select the output of GRU's hidden state of the last time step that contains all previous information as the final graph embedding $\boldsymbol{g} \in \mathbb{R}^d$, where d is the hidden layer dimension of GRU. Through the global temporal embedding extractor, the output $\boldsymbol{g}$ encompasses information on network evolutions. This furnishes distinguishable features for graph classification.

D. Network Training

TP-GNN is trained in an end-to-end manner. We first feed training data containing positive and negative samples into the model to learn the corresponding graph embedding $\boldsymbol{g}_i$. After that, we utilize a fully connected network $\varphi(\cdot)$ as a classifier to complete downstream graph classification. The output is the prediction label $\varphi(\boldsymbol{g}_i) \in \{0, 1\}$ for the graph $\mathcal{G}_i$, where the positive graph is labeled 1 and the negative graph is labeled 0. The loss function is formalized as follows:

$$\varphi(\boldsymbol{g}_i) = Sigmoid(\boldsymbol{W_e}\boldsymbol{g}_i) + \boldsymbol{b_e}), \quad (11)$$

$$\mathcal{L} = -(Y_{\mathcal{G}_i} log(\varphi(\boldsymbol{g}_i)) + (1 - Y_{\mathcal{G}_i}) log(1 - \varphi(\boldsymbol{g}_i)), \quad (12)$$

where $\boldsymbol{W_e}$ and $\boldsymbol{b_e}$ are the weights and bias parameters of the fully connected network layer, and $Y_{\mathcal{G}_i}$ is the ground truth label of graph $\mathcal{G}_i$.

E. Complexity Analysis

We next analyze the time complexity of TP-GNN. The complexity of the temporal propagation module depends on the message updating method. Give a dynamic network $\mathcal{G}$, in the temporal-propagation-SUM method, we process in timestamp order with time complexity of $O(mk)$, while the complexity in the temporal-propagation-GRU method is $O(mk^2)$, where m is the number of edges in each graph and k is the dimension of the node feature vector. The global temporal embedding extractor uses GRU to process all edges in $\mathcal{G}$, so the time complexity is $O(md^2)$, where d is the hidden size of GRU. In summary, the time complexities of the entire model with SUM-updater and GRU-updater are $O(mk + md^2)$ or $O(mk^2 + md^2)$, respectively. For one of the most competitive baselines, the theoretical time complexity of TGAT [46] is $O(b^K d^2)$, where b is the number of sampled neighbors, K is the number of TGAT layers, and d is the dimension of node embedding. It can be seen that with the increase of the number of layers, the exponential expansion of TGAT sampling neighbors will seriously affect the model efficiency. Our model updates node features only once per timestamp, thus it is more efficient when dealing with large-scale sparse dynamic graphs with many nodes.

V. Experiments

In this section, we conduct graph classification in dynamic networks to investigate the effectiveness of TP-GNN framework. We first describe the configurations of the baselines and our method in detail. Besides, the performance of the model in terms of ablation study, parameter sensitivity, and running time are comprehensively analyzed.

A. Datasets

We choose five datasets from various domains for the graph classification experiments to demonstrate the robustness of the proposed model under different realistic scenarios. Notably, the Forum-java dataset is created based on a publicly available java forum system by ourselves.

- **Forum-java**[2] is a log dataset that we collected from an open source java-based forum system [3]. Every log records a series of dynamic system behaviors generated during the operation of the forum system. We parse the log data based on the trace IDs into 172,443 dynamic session networks [33]. Each node in the network represents a log event and contains features about invoking information, duration, and exceptions. Each edge denotes the order in which log events occur. To generate negative graphs, we introduce four types of faults based on a real industry case into the system [4]. By running different faulty versions of the system, we generate the networks labeled as "negative".
- **HDFS** [40] is a publicly available log dataset stored in the HDFS distributed file system, which is widely used in the field of log anomaly detection. Similar to the Forum-java dataset, we parse the log data into 575,061 dynamic session networks, out of which 16,838 have been meticulously labeled as "negative" by domain experts specializing in log anomaly detection. The difference is that the partitioning of this dataset is based on the block IDs rather than the trace IDs in the log. For each node, we encode the log event's level, source module, and thread ID into original node features by using the label coding method.
- **Brightkite** [5], **Gowalla** [5] and **FourSquare** [43] are public user trajectory datasets used to detect if users' trajectories are abnormal. In these datasets, users share their location data by sending check-in events from their mobile devices. Brightkite, Gowalla, and FourSquare contain a total of 4,491,143 check-ins from 58,227 users, 6,442,890 check-ins from 75,887 users, and 33,278,683 check-ins from 266,909 users, respectively. To facilitate analysis, we divide

[2] *https://github.com/Jie-0828/TP-GNN/tree/main/Dataset/Forum-java.*
[3] *https://github.com/Qbian61/forum-java.*

the dataset into multiple dynamic networks based on user IDs. In each graph, nodes represent the check-in locations of users and edges represent the movement of the user's position. We take the values of the longitude, latitude, and country ID of the check-in location as the original node features. The dynamic graphs formed by the check-ins of normal users in the original datasets are marked "positive". The generation of "negative" examples is discussed in the following paragraph. The graph classification task is to detect whether there is abnormal user check-in behavior.

The HDFS, Gowalla, and Brightkite datasets are all publicly available datasets collected in the real world, they all lack available negative samples. Therefore, we first filter out graph samples with less than three records to delete inactive users and system requests. In addition, we utilize two methods for generating negative samples that differ either structurally or temporally from the positive samples. First, we adopt the "context-dependent" negative sampling strategy [2] to generate negative samples with different structures. Specifically, for a given positive sample, we randomly select a small number of edges in the graph. For each selected edge $e = (u, v, t)$, we replace one of the nodes to generate a new edge $e' = (u, v', t)$. If e' belongs to the normal graph, we delete it; Otherwise, we retain it as a negative edge. The dynamic network with negative edges generated according to the above process is defined as a negative sample with a different structure. On the other hand, we randomly shuffle the edge establishment order in the positive sample to generate a negative sample with different time sequences that differs from the normal dynamic graph. Table I shows the statistics of the pre-processed datasets.

B. Baselines

We select twelve representative models that are easily adaptable for the graph classification task as benchmarks. The initial four models correspond to static graph methods, the subsequent four are discrete DGNNs, and the final four are continuous DGNNs.

- **Spectral Clustering [20]**is a graph-theory-based clustering algorithm that treats data as nodes within static graphs. By maximizing the similarity between nodes in the neighborhood, the node embeddings are learned.
- **GCN** [14] is a classical static graph convolutional network. For each node in the graph, the model integrates the features of itself and all of its neighbors for feature updating.
- **GraphSage** [11] is an inductive static GNN. By selectively aggregating neighbor nodes, it is easy to extend the model to learn node features in graphs with more complex structures.
- **GAT** [32] is an attention mechanism-based static GNN that uses attention scores for weighted aggregation of neighbor node features to better capture correlations between nodes.
- **AddGraph [45]**is a discrete DGNN that leverages temporal GCN with an attention-based GRU to learn node features. As a semi-supervised learning framework, AddGraph employs selective negative sampling and a marginal loss function during training to improve the model performance.
- **TADDY [16]**is a transformer-based discrete DGNN. The model introduces a node coding to represent the temporal and structural information of nodes in discrete snapshots. Additionally, Transformer [31] is used to capture valuable node embeddings from snapshots.
- **EvolveGCN** [22] is a discrete DGNN that uses GCN and RNN respectively to extract spatial-temporal information. In order to ensure the reliability and stability of the model, we realize this baseline based on the EvolveGCN-H model encapsulated in Pytorch Geometric temporal [26].
- **GC-LSTM** [3] is also a discrete DGNN that utilizes LSTM as the primary framework for learning temporal dependencies between snapshots. Like EvolveGCN, we use the encapsulated GC-LSTM method in Pytorch geometric temporal [26] as the baseline.
- **TGAT** [39] is an advanced continuous DGNN for dynamic graph representation learning. The method uses the self-attention mechanism as a building block and a functional time encoding based on Bochner's theorem to capture the coupled spatial-temporal information in the network.
- **DyGNN** [18] is an prevailing continuous DGNNs. The model designs the update and propagation components to model the dynamic information as the graph evolves.
- **TGN** [25] is a general and efficient continuous DGNN framework for deep learning on dynamic graphs represented as sequences of time events. This framework learns node embedding through the message function, message aggregator, memory updater, and embedding modules.
- **GraphMixer** [7] is a state-of-the-art continuous DGNN. This model employs an MLP-based link encoder and a neighbor mean-pooling-based node encoder to capture the temporal information of edge and node features, thereby executing the link prediction task.

C. Evaluation metrics

Three mainstream indicators in graph classification – Precision, Recall, and F_1 Score are used to measure the effectiveness of the TP-GNN.

- **Precision**: the percentage of positive samples out of all samples detected as positive, represented as $precision = \frac{TP}{TP+FP}$.
- **Recall**: the percentage of all positive samples that are detected as positive, represented as $recall = \frac{TP}{TP+FN}$
- F_1 **Score**: the harmonic mean of precision and recall, represented as $F_1\ Score = \frac{2 \times Precision \times Recall}{Precision + Recall}$

D. Experimental Setup

Baselins Configuration: Firstly, for the static models – Spectral Clustering, GCN, GraphSage and GAT, we ignore the edge timestamps in datasets and treat data as static networks. For GraphSage, we choose the MEAN aggregator function to aggregate the neighbors' features. Corresponding to our model, the hidden layer size of all static models is set to 32. Secondly, for snapshot-based discrete DGNN baselines – AddGraph, TADDY, EvolveGCN, and GC-LSTM, we first crop every dataset into a series of static snapshots. The snapshot size is set

TABLE I: Key statistics of datasets used in experiments.

Datasets	Forum-java	HDFS	Gowalla	FourSquare	Brightkite
Graph Number	172,443	130,344	105,862	347,848	44,693
Negative ratio	∼32.5%	∼29.8%	∼28.8%	∼30.3%	∼30.3%
Avg # Node	27	12	72	61	46
Avg # Edge	30	31	117	135	188
# Node features	3	3	3	3	3

to 5 for Forum-java and HDFS datasets, and 20 for Gowalla and Brightkite datasets. Other hyperparameters are fine-tuned based on their original papers to achieve optimal performance across different datasets respectively. Besides, for continuous DGNNs including TGAT, DyGNN, TGN, and GraphMixer, these models have the capability to directly handle dynamic networks in a continuous manner without requiring additional processing on the datasets. For TGAT, we use two TGAT layers and two attention heads to give the best performance. For TGN, the attention head is set to 2, the memory dimension and the node embedding dimension to 32, and the time dimension to 6. For GraphMixer, we set the layer of MLP-Mixer to 2, and the time dimension to 6. For DyGNN, the best performance is achieved by setting the hyperparameters as default in [18]. Other hyperparameters of these models are chosen empirically, following guidance from literature. Moreover, since the current works only focus on node- or edge-level tasks, we use the $Mean$ graph pooling operation [38] to transform learned the node or edge representations into graph representations to complete the graph classification.

TP-GNN Configuration: We divided the first 30% graphs of each dataset for training and the last 70% for testing. By default, the GRU hidden size d of TP-GNN is 32, and the time dimension d_t is 6. We adopt the Adam optimizer with a learning rate of 0.001 for all datasets. Our model is trained for 10 epochs. To account for the multiple interactions between nodes in the same timestamp within the given datasets, our model shuffles the edge order at the same timestamp before each training. This approach effectively eliminates any instability in the model that may be caused by different edge orders at the same timestamp. All experiments and timings are conducted on a Windows server with Intel(R) Xeon(R) 2.20GHz CPU and Tesla V100 GPU, running with Python 3.9, PyTorch 1.12.0. All experimental results presented in this paper are the average values obtained after five runs of the model. The code will be made available for all our experiments to be reproduced.

E. Results on Dynamic Graph Classification

Table II shows the results of the dynamic graph classification task on five different datasets. We will analyze the experimental results from the following three aspects:

Compare with static models: Among the four static models (GCN, GraphSage, GAT, and Spectral Clustering), TP-GNN shows significant improvements in F_1 Score, Precision, and Recall on all baseline datasets. This is because static methods ignore the critical temporal information during message passing, so the accuracy of graph classification is seriously affected. In addition, since Spectral Clustering elies on the graph's Laplacian matrix in the calculation process, the graph data need to be treated as an undirected graph. Meanwhile, the Spectral Clustering also ignores the node features, so this approach exhibits the poorest performance on all datasets.

Compare with discrete DGNNs: Regarding discrete DGNNs – EvolveGCN, GC-LSTM, AddGraph, and TADDY, they take into account temporal information in the network. Therefore, the experimental results show that the F_1 Score has a certain increase compared with the static models. However, these models intuitively treat a dynamic network as a series of independent static snapshots, thus inevitably leading to a loss of temporal information within snapshots. This omission gives rise to learning inappropriate node embeddings, ultimately affecting the model's performance on downstream tasks. Hence, discrete DGNNs do not perform as well as continuous DGNNs, nor as well as our model on the graph classification task.

Compare with continuous DGNNs: In contrast to the four most competitive continuous DGNNs – TGN, DyGNN, TGAT, and GraphMixer, our model still achieves remarkable outcomes, particularly with an average improvement of 4.91% in term of F_1 Score. Notably, TP-GNN-GRU outperforms TP-GNN-SUM on large Brightkite and FourSquare datasets, attributed to the GRU-updater adeptness in capturing the evolutionary patterns of node features in longer interaction sequences. Moreover, on datasets with large-scale dynamic networks, our model shows a stronger advantage. The experimental results denote that TP-GNN outperforms the SOTA model by 9.86% in terms of F_1 Score on Brightkite dataset. This further underscores the potential of our model to comprehensively learn and capture data features during the training.

Overall, the reasons underlying TP-GNN's superiority over the baselines can be summarized as follows: 1)Our method can directly model dynamic networks, effectively circumventing the temporal information loss caused by processing discrete snapshots. 2) TP-GNN follows the direction of information flow to aggregate influential node features. This message-passing method extracts not only local spatial-temporal information but also long temporal dependencies in dynamic networks. 3) TP-GNN incorporates a global temporal embedding extractor for extracting the global temporal information, which more accurately simulates the dynamic evolution process of the network and also solves the problem of oversmoothing caused by multiple message passing. In short, both theoretical analysis and experimental results prove that our model has excellent performance in dynamic graph classification.

F. Ablation Study

In this section, ablation studies are conducted to evaluate the effects of each component of TP-GNN. For illustration purposes, we generate the following model variables:

- `rand`: In this variant, we use random aggregation in place of temporal propagation. In other words, this variant ignores the temporal information and randomly samples neighbor nodes for aggregation. Meanwhile, since we remove the global temporal embedding extractor, the $Mean$ graph pooling operation is used to transform the node representation into graph representation.

TABLE II: Average F_1 Score, Recall, and Precision for graph classification (%). The upper four baselines are static graph methods, the following four baselines are discrete DGNNs, and the last four baselines are continuous DGNNs. The best-performing method in each experiment is in bold.

	Forum-java			HDFS		
	F_1 Score	Precision	Recall	F_1 Score	Precision	Recall
Spectral Clustering	$74.23_{\pm 0.95}$	$71.20_{\pm 1.56}$	$77.52_{\pm 0.55}$	$61.71_{\pm 0.97}$	$70.84_{\pm 0.43}$	$54.66_{\pm 1.12}$
GCN	$83.86_{\pm 0.76}$	$72.85_{\pm 1.13}$	$98.80_{\pm 0.02}$	$84.49_{\pm 0.04}$	$73.26_{\pm 0.66}$	$99.81_{\pm 0.12}$
GraphSage	$84.11_{\pm 0.18}$	$84.72_{\pm 1.07}$	$81.76_{\pm 1.11}$	$86.60_{\pm 1.04}$	$77.59_{\pm 0.99}$	$99.36_{\pm 0.64}$
GAT	$80.12_{\pm 0.05}$	$67.45_{\pm 0.01}$	$98.65_{\pm 0.12}$	$82.91_{\pm 1.01}$	$77.72_{\pm 1.45}$	$99.48_{\pm 0.07}$
AddGraph	$84.67_{\pm 0.03}$	$74.22_{\pm 0.48}$	$98.54_{\pm 0.03}$	$87.20_{\pm 0.41}$	$80.88_{\pm 0.69}$	$94.60_{\pm 0.08}$
TADDY	$88.10_{\pm 0.05}$	$85.47_{\pm 0.73}$	$90.93_{\pm 0.94}$	$82.29_{\pm 0.10}$	$70.36_{\pm 0.01}$	$99.10_{\pm 0.32}$
EvolveGCN	$82.17_{\pm 0.48}$	$81.47_{\pm 1.13}$	$82.94_{\pm 2.15}$	$81.46_{\pm 0.13}$	$68.75_{\pm 0.20}$	$\mathbf{99.95}_{\pm 0.03}$
GC-LSTM	$87.67_{\pm 1.20}$	$84.39_{\pm 2.56}$	$91.55_{\pm 5.08}$	$89.71_{\pm 1.09}$	$81.65_{\pm 2.04}$	$99.57_{\pm 0.36}$
TGN	$93.12_{\pm 1.64}$	$97.95_{\pm 0.60}$	$88.84_{\pm 2.15}$	$89.54_{\pm 0.66}$	$84.10_{\pm 0.72}$	$95.79_{\pm 2.21}$
DyGNN	$94.25_{\pm 0.12}$	$95.60_{\pm 0.57}$	$92.94_{\pm 1.53}$	$94.89_{\pm 0.24}$	$91.89_{\pm 0.79}$	$98.10_{\pm 1.44}$
TGAT	$95.96_{\pm 0.15}$	$93.17_{\pm 0.44}$	$98.64_{\pm 0.58}$	$90.44_{\pm 1.20}$	$86.50_{\pm 3.60}$	$94.94_{\pm 1.71}$
GraphMixer	$96.44_{\pm 0.41}$	$96.95_{\pm 0.02}$	$95.95_{\pm 0.83}$	$93.06_{\pm 0.26}$	$89.55_{\pm 0.59}$	$96.87_{\pm 0.93}$
TP-GNN-GRU	$98.53_{\pm 0.33}$	$98.47_{\pm 0.68}$	$98.59_{\pm 0.06}$	$97.53_{\pm 0.18}$	$96.79_{\pm 0.38}$	$98.29_{\pm 0.31}$
TP-GNN-SUM	$\mathbf{99.21}_{\pm 0.15}$	$\mathbf{99.20}_{\pm 0.25}$	$\mathbf{99.24}_{\pm 0.44}$	$\mathbf{98.26}_{\pm 0.55}$	$\mathbf{98.17}_{\pm 1.98}$	$98.37_{\pm 0.94}$

	Gowalla			Foursquare			Brightkite		
	F_1 Score	Precision	Recall	F_1 Score	Precision	Recall	F_1 Score	Precision	Recall
Spectral Clustering	$58.47_{\pm 0.43}$	$71.11_{\pm 0.03}$	$49.71_{\pm 0.58}$	$63.41_{\pm 0.04}$	$74.70_{\pm 0.07}$	$55.25_{\pm 0.04}$	$62.63_{\pm 0.11}$	$70.63_{\pm 0.02}$	$56.26_{\pm 0.17}$
GCN	$82.90_{\pm 0.26}$	$73.73_{\pm 4.07}$	$98.24_{\pm 1.31}$	$82.10_{\pm 0.01}$	$71.31_{\pm 0.03}$	$96.74_{\pm 0.01}$	$76.56_{\pm 1.33}$	$69.54_{\pm 0.02}$	$85.22_{\pm 3.02}$
GraphSage	$83.21_{\pm 0.12}$	$72.54_{\pm 0.65}$	$97.57_{\pm 0.91}$	$83.11_{\pm 0.00}$	$74.70_{\pm 0.04}$	$94.33_{\pm 0.06}$	$80.12_{\pm 1.20}$	$69.67_{\pm 1.49}$	$94.26_{\pm 1.01}$
GAT	$87.76_{\pm 0.02}$	$78.99_{\pm 0.05}$	$98.71_{\pm 0.04}$	$81.75_{\pm 0.00}$	$69.63_{\pm 0.01}$	$98.99_{\pm 0.01}$	$81.42_{\pm 0.06}$	$69.82_{\pm 0.04}$	$97.65_{\pm 0.13}$
AddGraph	$82.82_{\pm 0.52}$	$71.47_{\pm 1.86}$	$98.45_{\pm 1.47}$	$85.59_{\pm 0.31}$	$76.87_{\pm 0.58}$	$96.52_{\pm 0.07}$	$81.31_{\pm 1.52}$	$68.51_{\pm 3.65}$	$99.98_{\pm 0.77}$
TADDY	$88.70_{\pm 0.45}$	$82.14_{\pm 1.98}$	$96.39_{\pm 1.16}$	$88.81_{\pm 0.00}$	$84.62_{\pm 0.02}$	$93.44_{\pm 0.01}$	$84.42_{\pm 1.02}$	$73.16_{\pm 0.98}$	$99.78_{\pm 1.63}$
EvolveGCN	$84.87_{\pm 0.59}$	$75.70_{\pm 2.04}$	$96.02_{\pm 1.31}$	$86.68_{\pm 0.00}$	$76.99_{\pm 0.00}$	$99.17_{\pm 0.01}$	$81.83_{\pm 0.24}$	$69.68_{\pm 0.01}$	$99.13_{\pm 0.72}$
GC-LSTM	$92.36_{\pm 0.32}$	$87.47_{\pm 0.23}$	$97.82_{\pm 0.87}$	$88.41_{\pm 0.01}$	$79.96_{\pm 0.025}$	$98.91_{\pm 0.01}$	$81.82_{\pm 0.10}$	$69.59_{\pm 0.02}$	$99.27_{\pm 1.04}$
TGN	$93.25_{\pm 0.16}$	$94.18_{\pm 0.15}$	$92.34_{\pm 0.49}$	$92.09_{\pm 0.01}$	$87.37_{\pm 0.05}$	$97.36_{\pm 0.04}$	$85.26_{\pm 0.04}$	$81.26_{\pm 0.92}$	$89.69_{\pm 0.05}$
DyGNN	$92.13_{\pm 0.04}$	$90.29_{\pm 0.06}$	$94.05_{\pm 0.07}$	$94.64_{\pm 0.00}$	$91.60_{\pm 0.03}$	$98.05_{\pm 0.03}$	$83.25_{\pm 0.04}$	$80.11_{\pm 0.13}$	$86.64_{\pm 0.01}$
TGAT	$91.96_{\pm 0.04}$	$87.30_{\pm 0.52}$	$97.15_{\pm 0.73}$	$91.89_{\pm 0.03}$	$86.44_{\pm 0.01}$	$98.99_{\pm 0.04}$	$84.57_{\pm 0.31}$	$73.73_{\pm 0.63}$	$99.14_{\pm 0.45}$
GraphMixer	$94.62_{\pm 0.02}$	$94.16_{\pm 0.17}$	$95.08_{\pm 0.02}$	$94.11_{\pm 0.00}$	$93.85_{\pm 0.01}$	$93.87_{\pm 0.00}$	$86.80_{\pm 0.10}$	$79.85_{\pm 0.43}$	$95.08_{\pm 0.62}$
TP-GNN-GRU	$98.08_{\pm 0.10}$	$\mathbf{97.37}_{\pm 0.48}$	$97.80_{\pm 0.64}$	$\mathbf{99.58}_{\pm 0.26}$	$\mathbf{99.27}_{\pm 0.55}$	$\mathbf{99.88}_{\pm 0.59}$	$\mathbf{96.66}_{\pm 0.08}$	$\mathbf{95.05}_{\pm 0.49}$	$98.33_{\pm 0.59}$
TP-GNN-SUM	$\mathbf{98.23}_{\pm 0.29}$	$96.76_{\pm 0.66}$	$\mathbf{99.76}_{\pm 0.21}$	$99.02_{\pm 0.00}$	$98.72_{\pm 0.00}$	$99.32_{\pm 0.00}$	$95.61_{\pm 0.24}$	$91.58_{\pm 0.44}$	$\mathbf{100.00}_{\pm 0.00}$

- `w/o tem`: In this variant, we remove the temporal propagation and solely rely on the global temporal embedding extractor. Specifically, we transform the initial node features into edge embeddings and feed them into GRU according to the timestamped edges to capture the dynamic evolution of the network.
- `temp`: In this variant, we remove the global temporal embedding extractor. Besides, the time embedding vector is eliminated from the temporal propagation. After temporal propagation, we also use the $Mean$ graph pooling layer to get the graph embeddings.
- `time2Vec`: In this variant, we solely remove the global temporal embedding extractor from the full model. This variant is used to demonstrate the validity of the time embedding vector in ablation studies. This variant obtains the graph embedding by the $Mean$ graph pooling layer.

The validity of temporal propagation: The temporal propagation mainly comprises the time embedding vector and message-passing mechanism based on information flow. We will show how they affect the performance of the model. The ablation studies results of TP-GNN-SUM and TP-GNN-GRU are illustrated in Figure 3 and Figure 4, respectively.

Firstly, we explore the role of the message-passing mechanism. The experimental results show that on Form-java and HDFS datasets, compared with that of `rand` variant, the `temp` variant is improved in terms of F_1 Score, Precision, and Recall. Since each graph in Brightkite dataset contains more nodes and edges, multiple messages may cause oversmoothing, leading to information loss. Thus the improvement is less effective than other datasets. However, careful observation still shows that the message-passing mechanism that respects the direction of information flow can effectively capture the long temporal dependencies in the network, thereby improving model performance. In addition, due to the good performance of the GRU-updater in capturing the evolution of node features, on most datasets, the `temp` variant of TP-GNN-GRU shows better performance than the `temp` variant of TP-GNN-SUM in the absence of the global temporal embedding extractor. Secondly, to demonstrate the influence of the time embedding vector (i.e., $\boldsymbol{f(t)}$) on model performance, we use the `time2Vec` variant with $\boldsymbol{f(t)}$ for illustration. The experimental results show that the `time2Vec` variant garners a higher F_1 Score on all datasets than the `temp` variant without the $\boldsymbol{f(t)}$. It is seen that in the process of message propagation, aggregation of time embedding vectors representing interaction time can effectively enhance the performance of TP-GNN.

TABLE III: Average F_1 Score (%) of different methods using global temporal embedding extractor for graph classification.

Datasets	Forum-java	HDFS	Gowalla	Brightkite
TGAT+G	97.87	95.14	94.33	93.65
DyGNN+G	97.12	97.87	95.93	94.90
TGN+G	97.65	97.17	93.50	92.38
GraphMixer+G	98.04	96.62	96.25	94.23
TP-GNN-SUM	**99.21**	**98.16**	**98.23**	95.61
TP-GNN-GRU	98.27	97.52	97.42	**96.66**

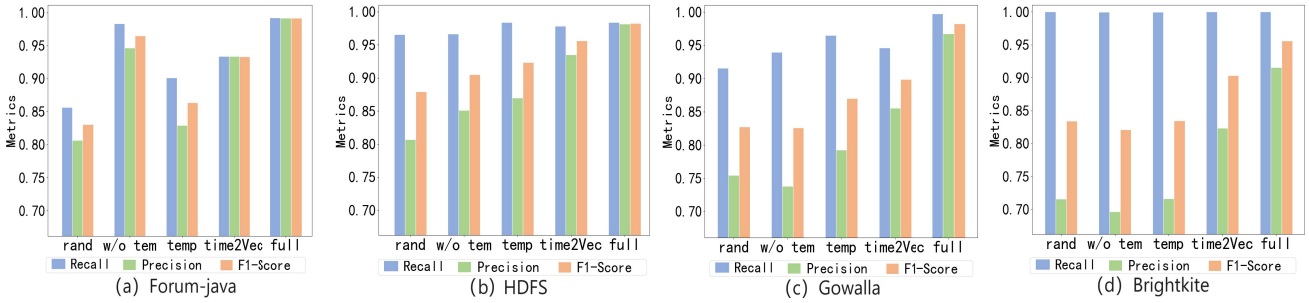

Fig. 3: Ablation Studies of TP-GNN-SUM on (a)Forum-java; (b)HDFS; (c)Gowalla; and (d)Brightkite datasets.

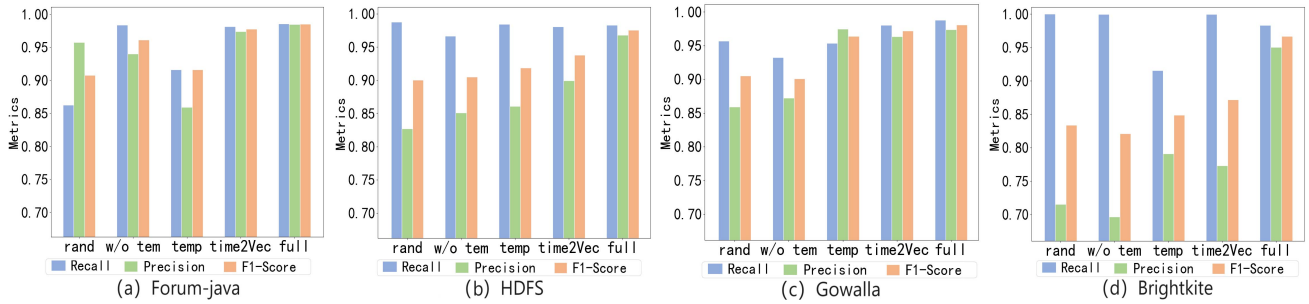

Fig. 4: Ablation Studies of TP-GNN-GRU on (a)Forum-java; (b)HDFS; (c)Gowalla; and (d)Brightkite datasets.

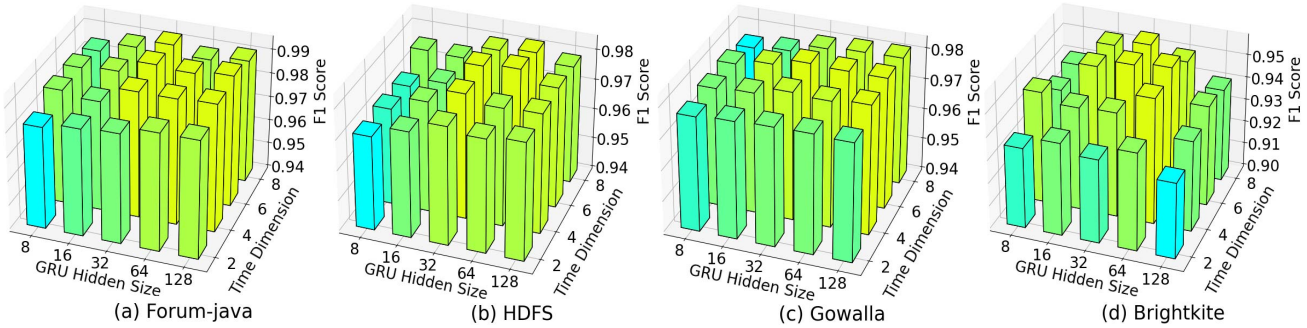

Fig. 5: A comparison of F_1 Score under different parameter settings (GRU hidden size d and time dimension d_t) on (a)Forum-java; (b)HDFS; (c)Gowalla ; (d)Brightkite datasets. The vertical axis represents the F_1 Score of TP-GNN with different d and d_t. A lighter color indicates a higher F_1 Score.

To further prove the superiority of temporal propagation, we compare the `full` model with the `w/o tem` variant. The experimental results show that no matter TP-GNN-SUM or TP-GNN-GRU, the F_1 Score of the `full` model is always higher than that of the `w/o tem` variant. Meanwhile, we conducted a series of experiments with continuous DGNNs instead of temporal propagation. The experimental results in Table III illustrate that our temporal propagation method outperforms other continuous DGNNs. This further confirms that, in comparison to existing message-passing mechanisms in continuous DGNNs, temporal propagation is adept at capturing long temporal dependencies within dynamic networks. Such information helps to learn appropriate node embeddings and provides valuable background information for the subsequent global temporal embedding extractor, thus improving the accuracy of dynamic graph classification.

The validity of global temporal embedding extractor: Global temporal information can effectively reflect the whole temporal evolution process of the network topology, which plays a key role in fully displaying the dynamic changes of the interaction between systems' entities and realizing the graph classification. The results in Fig. 3 and Fig. 4 indicate that the `full` model outperforms the `time2Vec` variant in terms of F_1 Score on all datasets. This shows that the global temporal embedded extractor can effectively extract the global temporal information from the order of edge establishment, thus improving the model performance.

G. Hyperparameter Sensitivity Analysis

In the hyperparameter sensitivity analysis, we focus on investigating the impact of the GRU hidden size d and the time dimension d_t on TP-GNN-SUM. We conduct experiments on $d \in \{8, 16, 32, 64, 128\}$ and $d_t \in \{2, 4, 6, 8\}$. Regarding the hyperparameter d and d_t, the experimental results in Fig. 5

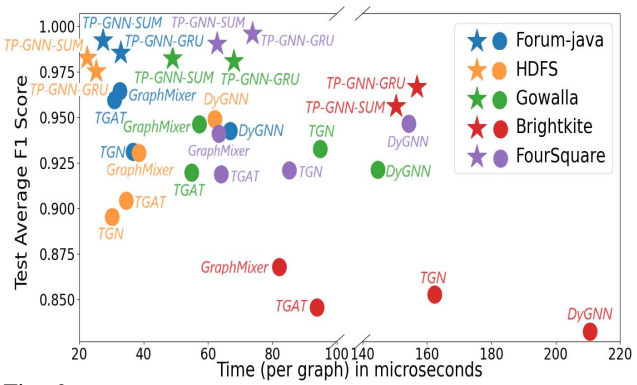

Fig. 6: Running times (per graph in microseconds) of different continuous dynamic graph neural networks on four datasets. Each dataset is represented by a different color. Meanwhile, "⋆" represents our proposed model TP-GNN, "•" represents the baselines. Here, the model closer to the top-left corner performs better in both running time and F_1 Score for graph classification.

show that with the increase of d and d_t, the F_1 Score gradually increases and then becomes stable. The highest performance of TP-GNN is achieved when d= 32, d_t=6 on all datasets. It is proved that when the hyperparameter is small, the model may miss useful information. This phenomenon is more evident on Brightkite dataset. It is possibly because the small sample size of the dataset affects the stability of the model and makes the model more sensitive to changes in the hyperparameters. Overall, our model exhibits superior performance across various hyperparameter settings, indicating its robustness. Considering the excellent performance of TP-GNN when the GRU hidden size is 32 and the time dimension is 6, we configured this as the default setting for all experiments.

To analyze the operating efficiency of our model, we conduct a comparative analysis of the running time performance of TP-GNN alongside four mainstream continuous DGNNs. The experimental results are depicted in Fig. 6. Since DyGNN involves two processes of node feature updating and interactive information propagation based on LSTM, it presents the longest running time on all datasets. Notably, except on the Brightkite dataset, our model outperforms other continuous DGNNs in both running time and F_1 Scores. This is likely because each graph in the Brightkite dataset is a dense graph with a large number of edges, and the time complexity of the temporal propagation and the global temporal embedding extractor of TP-GNN are proportional to the number of edges. Although our model necessitates longer training time for larger datasets compared to MLP-based GraphMixer and neighborhood-based and TGAT, it shows a marked improvement in F_1 Score. Overall, TP-GNN shows excellent efficiency and performance for the graph classification task on dynamic graph datasets of different sizes, which indicates its broad applicability.

H. Case Study

To demonstrate that the proposed TP-GNN can effectively capture long temporal dependencies and improve the graph classification performance of the model, we design a case

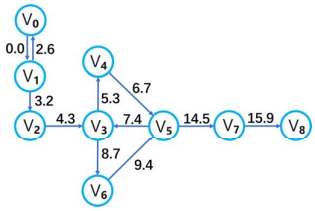

Fig. 7: A dynamic user-trajectory network from the Brightkite dataset.

study. In particular, we first select a positive dynamic user-trajectory network (See Fig. 7) from the Brightkite dataset. The network represents a user's behavior trajectory, with a node representing a POI and an edge representing the movement of the user's POI. In Fig. 7, when we exchange the edge between node v_2 and v_3 at t=4.3 with the edge between node v_5 and v_7 at t=14.5, TP-GNN will recognize the modified graph as a negative sample. According to the analysis, when the information flow of the dynamic graph changes, the node features learned by the model will change. For example, node v_7 at $t = 14.5$ in the positive graph will aggregate all node features except node v_8. When the information flow is changed, node v_7 will only aggregate the features of v_5, and the long temporal dependencies from v_0 to v_6 will be lost. Likewise, when modifying the direction of the edge between nodes v_5 and v_7 at t=14.5, the model is also able to recognize it efficiently. This shows that TP-GNN can distinguish different graphs by information flow in dynamic graphs, and improve the accuracy of graph classification task.

VI. Conclusion

In this paper, we study graph classification on dynamic networks for the first time and propose TP-GNN, a simple and effective continuous dynamic graph neural network. Firstly, to make up for the limitation of current DGNNs capturing local spatial-temporal information by aggregating h-hop neighbor nodes, TP-GNN designs the temporal propagation method to extract long temporal dependencies according to the direction of information flow in the network. Secondly, to capture the overall temporal evolution of dynamic networks more accurately, we propose a GRU-based global temporal embedding extractor, which effectively improves the classification performance of the model. Finally, we perform dynamic graph classification on five datasets, including one dataset created by ourselves. Experimental results show that TP-GNN outperforms all baselines. The hyperparameter sensitivity and running time analysis of the model also validate the robustness and high efficiency of TP-GNN on datasets of different sizes. In the future, for larger-size graphs, we will design a higher-performing time series model to capture global temporal information. In addition, we will consider more complex and challenging datasets, and design a suitable unsupervised model for the graph classification task.

Acknowledgment

The author would like to express their sincere thanks to the reviewers for their careful review and constructive comments.

REFERENCES

[1] Akshay Badola, Vineet Padmanabhan Nair, and Rajendra Prasad Lal. An analysis of regularization methods in deep neural networks. In *2020 IEEE 17th India Council International Conference (INDICON)*, pages 1–6, 2020.

[2] Lei Cai, Zhengzhang Chen, Chen Luo, Jiaping Gui, Jingchao Ni, Ding Li, and Haifeng Chen. Structural temporal graph neural networks for anomaly detection in dynamic graphs. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, CIKM '21, page 3747–3756, New York, NY, USA, 2021. Association for Computing Machinery.

[3] Jinyin Chen, Xueke Wang, and Xuanheng Xu. Gc-lstm: Graph convolution embedded lstm for dynamic network link prediction. *Applied Intelligence*, 52(7):7513–7528, 5 2022.

[4] Zimin Chen, Steve Kommrusch, Michele Tufano, Louis-Noël Pouchet, Denys Poshyvanyk, and Martin Monperrus. Sequencer: Sequence-to-sequence learning for end-to-end program repair. *IEEE Transactions on Software Engineering*, 47(9):1943–1959, 2019.

[5] Eunjoon Cho, Seth A. Myers, and Jure Leskovec. Friendship and mobility: User movement in location-based social networks. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, page 1082–1090, New York, NY, USA, 2011. Association for Computing Machinery.

[6] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[7] Weilin Cong, Si Zhang, Jian Kang, Baichuan Yuan, Hao Wu, Xin Zhou, Hanghang Tong, and Mehrdad Mahdavi. Do we really need complicated model architectures for temporal networks? *International Conference on Learning Representations (ICLR)*, 2023.

[8] Youcef Djenouri, Asma Belhadi, Jerry Chun-Wei Lin, Djamel Djenouri, and Alberto Cano. A survey on urban traffic anomalies detection algorithms. *IEEE Access*, 7:12192–12205, 2019.

[9] Daniel M. Dunlavy, Tamara G. Kolda, and Evrim Acar. Temporal link prediction using matrix and tensor factorizations. *ACM Transactions on Knowledge Discovery from Data*, 5(2):1–27, feb 2011.

[10] Chao Gao, Junyou Zhu, Fan Zhang, Zhen Wang, and Xuelong Li. A novel representation learning for dynamic graphs based on graph convolutional networks. *IEEE Transactions on Cybernetics*, 53(6):3599–3612, 2023.

[11] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[12] Bernard J Jansen. Search log analysis: What it is, what's been done, how to do it. *Library & information science research*, 28(3):407–432, 2006.

[13] Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart, and Marcus A. Brubaker. Time2vec: Learning a vector representation of time. *CoRR*, abs/1907.05321, 2019.

[14] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations (ICLR)*, 2017.

[15] Guohao Li, Chenxin Xiong, Guocheng Qian, Ali Thabet, and Bernard Ghanem. Deepergcn: Training deeper gcns with generalized aggregation functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(11):13024–13034, 2023.

[16] Yixin Liu, Shirui Pan, Yu Guang Wang, Fei Xiong, Liang Wang, Qingfeng Chen, and Vincent CS Lee. Anomaly detection in dynamic graphs via transformer. *IEEE Transactions on Knowledge and Data Engineering*, 35(12):12081–12094, 2023.

[17] Antonio Longa, Veronica Lachi, Gabriele Santin, Monica Bianchini, Bruno Lepri, Pietro Lio, Franco Scarselli, and Andrea Passerini. Graph neural networks for temporal graphs: State of the art, open challenges, and opportunities. *Transactions on Machine Learning Research*, 2023.

[18] Yao Ma, Ziyi Guo, Zhaocun Ren, Jiliang Tang, and Dawei Yin. Streaming graph neural networks. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 719–728, 2020.

[19] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, et al. Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, volume 19, pages 4739–4745, 2019.

[20] Andrew Ng, Michael Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 14, 2001.

[21] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. Continuous-time dynamic network embeddings. In *Proceedings of the ACM Web Conference 2018*, WWW '18, pages 969–976, 2018.

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

[23] Liang Qu, Huaisheng Zhu, Qiqi Duan, and Yuhui Shi. Continuous-time link prediction via temporal dependent graph neural network. In *Proceedings of The Web Conference 2020*, WWW '20, page 3026–3032. Association for Computing Machinery, 2020.

[24] Giulio Rossetti and Rémy Cazabet. Community discovery in dynamic networks: a survey. *ACM computing surveys (CSUR)*, 51(2):1–37, 2018.

[25] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. *ICML 2020 Workshop on Graph Representation Learning*, 2020.

[26] Benedek Rozemberczki, Paul Scherer, Yixuan He, George Panagopoulos, Alexander Riedel, Maria Astefanoaei, Oliver Kiss, Ferenc Beres, Guzmán López, Nicolas Collignon, et al. Pytorch geometric temporal: Spatiotemporal signal processing with neural machine learning models. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 4564–4573, 2021.

[27] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao 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 (WSDM)*, WSDM '20, page 519–527, New York, NY, USA, 2020. Association for Computing Machinery.

[28] Joakim Skarding, Bogdan Gabrys, and Katarzyna Musial. Foundations and modeling of dynamic networks using dynamic graph neural networks: A survey. *IEEE Access*, 9:79143–79168, 2021.

[29] Sheng Tian, Jihai Dong, Jintang Li, Wenlong Zhao, Xiaolong Xu, Baokun Wang, Bowen Song, Changhua Meng, Tianyi Zhang, and Liang Chen. Sad: Semi-supervised anomaly detection on dynamic graphs. In Edith Elkind, editor, *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2306–2314, 8 2023.

[30] Rakshit S. Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. Dyrep: Learning representations over dynamic graphs. In *International Conference on Learning Representations (ICLR)*, 2019.

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[32] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, 2018.

[33] Yi Wan, Yilin Liu, Dong Wang, and Yujin Wen. Glad-paw: Graph-based log anomaly detection by position aware weighted graph attention network. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 66–77. Springer, 2021.

[34] Xuhong Wang, Ding Lyu, Mengjian Li, Yang Xia, Qi Yang, Xinwen Wang, Xinguang Wang, Ping Cui, Yupu Yang, Bowen Sun, and Zhenyu Guo. Apan: Asynchronous propagation attention network for real-time temporal graph embedding. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, page 2628–2638, New York, NY, USA, 2021. Association for Computing Machinery.

[35] Yanbang Wang, Yen-Yu Chang, Yunyu Liu, Jure Leskovec, and Pan Li. Inductive representation learning in temporal networks via causal anonymous walks. *International Conference on Learning Representations (ICLR)*, 2021.

[36] Zhihao Wen and Yuan Fang. TREND: TempoRal event and node dynamics for graph representation learning. In *Proceedings of the ACM Web Conference 2022*, WWW '22. ACM, apr 2022.

[37] Lingfei Wu, Peng Cui, Jian Pei, Liang Zhao, and Xiaojie Guo. Graph neural networks: foundation, frontiers and applications. In *Proceedings*

of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, pages 4840–4841, 2022.

[38] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021.

[39] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. *International Conference on Learning Representations (ICLR)*, 2020.

[40] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan. Largescale system problem detection by mining console logs. *Proceedings of SOSP'09*, 2009.

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

[42] Lejing Yan, Chao Luo, and Rui Shao. Discrete log anomaly detection: A novel time-aware graph-based link prediction approach. *Information Sciences*, 647:119576, 2023.

[43] Dingqi Yang, Daqing Zhang, and Bingqing Qu. Participatory cultural mapping based on collective behavior data in location-based social networks. *ACM Transactions on Intelligent Systems and Technology*, 7(3), jan 2016.

[44] Yao Zhang, Yun Xiong, Yongxiang Liao, Yiheng Sun, Yucheng Jin, Xuehao Zheng, and Yangyong Zhu. Tiger: Temporal interaction graph embedding with restarts. In *Proceedings of the ACM Web Conference 2023*, WWW '23, page 478–488, New York, NY, USA, 2023. Association for Computing Machinery.

[45] Li Zheng, Zhenpeng Li, Jian Li, Zhao Li, and Jun Gao. Addgraph: Anomaly detection in dynamic graph using attention-based temporal gcn. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, IJCAI'19, page 4419–4425. AAAI Press, 2019.

[46] Tongya Zheng, Xinchao Wang, Zunlei Feng, Jie Song, Yunzhi Hao, Mingli Song, Xingen Wang, Xinyu Wang, and Chun Chen. Temporal aggregation and propagation graph neural networks for dynamic representation. *IEEE Transactions on Knowledge and Data Engineering*, 35(10):10151–10165, oct 2023.

[47] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.

[48] Yingzheng Zhu, Xiufang Liang, Huajuan Duan, Fuyong Xu, Yuanying Wang, Peiyu Liu, and Ran Lu. Node representation learning with graph augmentation for sequential recommendation. *Information Sciences*, 646:119405, 2023.