Meta-Learned Metrics over Multi-Evolution Temporal Graphs

Dongqi Fu*
University of Illinois at Urbana-Champaign
Illinois, USA
dongqif2@illinois.edu

Liri Fang*
University of Illinois at Urbana-Champaign
Illinois, USA
lirif2@illinois.edu

Ross Maciejewski
Arizona State University
Arizona, USA
rmacieje@asu.edu

Vetle I. Torvik
University of Illinois at Urbana-Champaign
Illinois, USA
vtorvik@illinois.edu

Jingrui He
University of Illinois at Urbana-Champaign
Illinois, USA
jingrui@illinois.edu

ABSTRACT

Graph metric learning methods aim to learn the distance metric over graphs such that similar (e.g., same class) graphs are closer and dissimilar (e.g., different class) graphs are farther apart. This is of critical importance in many graph classification applications such as drug discovery and epidemics categorization. Most, if not all, graph metric learning techniques consider the input graph as static, and largely ignore the intrinsic dynamics of temporal graphs. However, in practice, a graph typically has heterogeneous dynamics (e.g., microscopic and macroscopic evolution patterns). As such, labeling a temporal graph is usually expensive and also requires background knowledge. To learn a good metric over temporal graphs, we propose a temporal graph metric learning framework, Temp-GFSM. With only a few labeled temporal graphs, Temp-GFSM outputs a good metric that can accurately classify different temporal graphs and be adapted to discover new subspaces for unseen classes. Each proposed component in Temp-GFSM answers the following questions: What patterns are evolving in a temporal graph? How to weigh these patterns to represent the characteristics of different temporal classes? And how to learn the metric with the guidance from only a few labels? Finally, the experimental results on real-world temporal graph classification tasks from various domains show the effectiveness of our Temp-GFSM.

CCS CONCEPTS

• **Mathematics of computing** → **Graph algorithms**; • **Computing methodologies** → **Learning latent representations**.

KEYWORDS

Meta-Learning; Metric Learning; Temporal Graph Classification

*Both authors contributed equally to this research.

KDD '22, August 14–18, 2022, Washington, DC, USA

ACM ISBN 978-1-4503-9385-0/22/08…$15.00
https://doi.org/10.1145/3534678.3539313

ACM Reference Format:
Dongqi Fu, Liri Fang, Ross Maciejewski, Vetle I. Torvik, and Jingrui He. 2022. Meta-Learned Metrics over Multi-Evolution Temporal Graphs. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3534678.3539313

1 INTRODUCTION

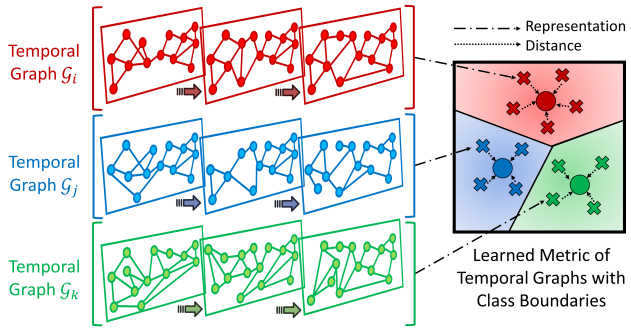


Figure 1: Metric Learning on Temporal Graphs $\mathcal{G}_i$, $\mathcal{G}_j$, $\mathcal{G}_k$, etc., from 3 Classes. In the right box (i.e., metric space), each 'X' denotes the representation of a single temporal graph, and each 'O' denotes the representation of a class.

Metric learning aims to learn a proper distance metric among data items in the input space, which reflects their underlying relationship. With the prevalence of graph data in many real-world applications, it is of key importance to design a good distance metric function for graph data, such that the output is small for similar (e.g., same class) graphs and large for dissimilar (e.g., different class) ones, as shown in Figure 1. Many downstream tasks on the graph data can benefit from such a graph metric. For example, it could lead to significantly improved classification accuracy for graph classification in many domains such as protein and drug discovery [11, 46], molecular property prediction [15, 23], and epidemic infectious pattern analysis [12, 41].

Most graph metric learning methods [4, 37, 47, 52, 67] assume the input graph data as static and ignore the evolution patterns of temporal graphs [17, 19–21, 62, 71, 72], which can also provide insights for identifying the graph property. To bridge this gap, we aim to learn a good metric over temporal graphs such that it is *accurate* (e.g., high temporal graph classification accuracy) and *comprehensive* (i.e., coordinating and adapting numerous seen and unseen

classes appropriately in the space). Several metric learning works have been proposed for i.i.d data [2, 25, 42, 45, 50, 55]. However, the relational data such as graphs especially the time-evolving ones challenge metric learning in the following aspects: (1) When a graph evolves, different evolution patterns are observed from different angles, e.g., microscopic dynamics from continuous time and macroscopic dynamics from discrete time [30, 35, 36]. What patterns and their weights are dominating the temporal graph similarity and largely influencing the metric space formation? (2) Because labeling graph data is typically expensive and requires background knowledge [27, 28, 43, 76], each class may only have a few labeled graphs, and some new classes may appear at a later time. How could we leverage a few labeled graphs to learn an accurate metric, which could accommodate the arrival of future unseen classes?

To answer these questions, we propose the Temp-GFSM framework to learn a good metric over temporal graphs. Temp-GFSM is an end-to-end trainable framework, which consists of three components: (1) *Multi-Time Evolution* modeling, which preserves multi-scope evolution patterns of a single temporal graph for the downstream metric learning process; (2) *Multi-Time Attention* modeling, which receives the output from the previous component, and assigns different weights to different evolution patterns in order to improve the effectiveness of the learned metric; (3) *Temporal Graph Few-Shot Metric Learning* process with the bi-level optimization. With Temp-GFSM, on the one hand, the learned metric will be accurate for the classes observed so far; on the other hand, it could be fast adapted to unseen classes to ensure metric comprehensiveness. Tuning the whole metric learning process is based on the loss of mis-classification, with only a few labels needed.

Our main contributions can be summarized as follows.

- We define the problem of temporal graph metric learning, and identify the challenges from real-world temporal graph data.
- We propose the temporal graph metric learning framework, Temp-GFSM, which addresses three challenges: (1) model a temporal graph with multiple dynamics, (2) weigh various evolution patterns, and (3) leverage a few labels to make the learned metric accurate and comprehensive.
- We test the accuracy and comprehensiveness of the metric learned by Temp-GFSM through few-shot temporal graph classification tasks from the biological network domain and social network domain, comparing with state-of-the-art baselines. We also demonstrate the convergence speed, the parameter sensitivity, and the ablation studies of Temp-GFSM.

The rest of the paper is organized as follows. We define the problem of temporal graph metric learning and review necessary preliminaries and corresponding challenges in Section 2. To solve the identified challenges, we propose a generic framework Temp-GFSM, with the detailed introduction of each component in Section 3. Then, in Section 4, we demonstrate the effectiveness of the metric learned by our Temp-GFSM through experiments. Finally, we conclude the paper in Section 6 after reviewing related work in Section 5. In this paper, we use "graph" and "network" interchangeably.

2 PROBLEM DEFINITION AND CHALLENGES

First of all, we formally describe the problem of *Learning Metrics over Temporal Graphs* as follows.

Problem. *Learning Metrics over Temporal Graphs*

Input: *a set of n temporal graphs* $\mathcal{D} = \{\mathcal{G}_1, \ldots, \mathcal{G}_n\}$.

Output: *the metric space* $\mathbb{M}$ *parameterized by* θ*, where similar (e.g., same class) temporal graphs are closer and dissimilar (e.g., different class) graphs are farther apart.*

To solve the above problem, our desired metric $\mathbb{M}$ should be both ***accurate***, i.e., making similar graphs close and dissimilar ones far apart, and ***comprehensive***, i.e., being able to handle not only the classes seen so far, but also new classes appearing at a later time. To achieve these goals, we first review preliminaries (Subsection 2.1) and then identify key challenges (Subsections 2.2 and 2.3).

2.1 Preliminary

According to [24, 45], learning a metric over a bunch of data items is closely related to the problem of extracting their hidden representation vectors. To be specific, given a metric $\mathbb{M}$, we can measure the distance $\mathbb{M}(\mathbf{x}_i, \mathbf{x}_j)$ between two input feature vectors $\mathbf{x}_i \in \mathbb{R}^m$ and $\mathbf{x}_j \in \mathbb{R}^m$ by computing $\mathbb{M}'(f_\theta(\mathbf{x}_i), f_\theta(\mathbf{x}_i))$, where f_θ is a learnable function mapping the input feature $\mathbf{x}_i$ into the latent feature $\mathbf{h}_i = f_\theta(\mathbf{x}_i) \in \mathbb{R}^h$. An example is shown below.

$$\begin{aligned}\mathbb{M}(\mathbf{x}_i, \mathbf{x}_j) &= \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^\top \mathbf{M} (\mathbf{x}_i - \mathbf{x}_j)} \\ &= \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^\top \mathbf{W}^\top \mathbf{W} (\mathbf{x}_i - \mathbf{x}_j)} \\ &= \sqrt{(\mathbf{W}\mathbf{x}_i - \mathbf{W}\mathbf{x}_j)^\top (\mathbf{W}\mathbf{x}_i - \mathbf{W}\mathbf{x}_j)} \\ &= \mathbb{M}'(f_\theta(\mathbf{x}_i), f_\theta(\mathbf{x}_j))\end{aligned} \tag{1}$$

where $\mathbf{M}$ is a symmetric positive semi-definite matrix in the Mahalanobis metric $\mathbb{M}$, and it can be decomposed by a certain matrix $\boldsymbol{W}$ as $\mathbf{M} = \mathbf{W}^\top \mathbf{W}$. Then, learning the metric $\mathbb{M}$ (e.g., Mahalanobis) on input features is equivalent to learning hidden features with a fixed metric $\mathbb{M}'$ (e.g., Euclidean) [50, 57]. f_θ can take a variety of forms. In addition to the linear transformation shown in Eq. 1, f_θ can also correspond to a non-linear transformation. In this way, it is able to model the higher-order correlations between input data dimensions than linear transformations [45, 50, 57].

Based on the above analysis, we are ready to introduce our temporal graph metric learning problem: learning a metric $\mathbb{M}$ over pairs of temporal graphs is to learn a function f_θ mapping temporal graphs to their representation vectors in the Euclidean space, such that similar (or same class) temporal graphs are closer and dissimilar (or different class) graphs are farther apart. In the rest of this paper, we refer to this objective as ***learning a metric*** $\mathbb{M}$ ***(or transformation*** f_θ***) controlled by parameters*** θ.

To reduce the computational complexity, learning metric $\mathbb{M}$ with labels can be scaled by involving the class representation concept (e.g., chunklet in [5] and prototype in [50]), such that a sample only need to be close to its class representation and far from other class representations in the learned metric space. In the rest of the paper, we refer to class representations as prototypes, and will introduce how the prototypes get generated in our Temp-GFSM.

2.2 Label Scarcity and Meta-Learning

As discussed in the previous subsection, the learning metric $\mathbb{M}$ is controlled by θ, a set of finite parameters. Next we use the supervised temporal graph classification task as an example to

view the utility of the metric $\mathbb{M}$ as in Figure 1, where the nearest class prototype will dictate the graph label. It can be formalized as $\mathbb{P}(\hat{y} = \mathcal{G} \mid \theta) = \mathbb{P}(\hat{y} = \mathcal{G} \mid \theta, \mathcal{D})$, which means any prediction $\hat{y}$ of temporal graphs is independent of the observed data $\mathcal{D}$ given the parameters θ. More specifically, we have the following claim.

Claim 1. *The parameters of parametric models are capturing everything to know about the data that is relevant for predictions [22].*

Basically, Claim 1 suggests that the ideal θ should capture everything about the data $\mathcal{D}$ needed for making future predictions. For our problem, this means that the ideal θ should be able to distinguish similar and dissimilar temporal graphs (**accuracy**). However, labeling temporal graph data is typically expensive and requires background knowledge [27, 28, 43, 76]. Therefore, in data $\mathcal{D}$, each class may contain only a few labeled temporal graphs $\mathcal{G}$. Even challenging, some classes may be absent during the training process but appear at a later time. So θ should also enable the model to have good generalization capabilities to accommodate future unseen classes (**comprehensive**). To achieve these goals, we aim to answer the following questions.

(C.1) How could we ensure the accuracy of $\mathbb{M}$ when the learning process could not leverage a large amount of labeled data?
(C.2) How could we adjust the learned metric space $\mathbb{M}$ to a new subspace to accommodate newly arrived classes while maintaining the acceptable margin for all existing classes?

Therefore, we formulate the problem of learning the metric $\mathbb{M}$ (i.e., f_θ) into a bi-level meta-learning paradigm [16] such that: (1) given few-shot labeled temporal graph data of each class, the learned metric $\mathbb{M}$ should accurately assign each observed graph into correct classes; (2) when a new (i.e., unseen) class arrives, previous learned metric $\mathbb{M}$ should be fast adapted (by updating to a new subspace for the new class's boundary) to achieve the comprehensiveness of the graph metric and to maintain the utility (e.g., classification accuracy). The details of the proposed bi-level optimization can be found in Subsection 3.4.

2.3 Evolution Patterns of Temporal Graphs

In addition to the questions in the last subsectioin, another key aspect is the modeling of the evolution patterns of temporal graphs (i.e., what dynamics features of graphs are fed into the learning process), which brings the following two additional questions.

(C.3) What are evolving in the input temporal graphs?
(C.4) Which evolution patterns are dominating the similarity (e.g., class labels) of input temporal graphs?

For (C.3), different studies gave different answers from different angles, such as discrete time and continuous time [30], macroscopic evolution and microscopic evolution [35, 36]. This phenomenon reminds us that even for the same temporal graph, the evolution patterns are different if we view it differently, which requires us to design a comprehensive model that could provide exhaustive dynamics information for the downstream learning process (e.g., selection and extraction). Therefore, we provide the *Multi-Time Evolution* model, which jointly models continuous time to observe microscopic evolution and discrete time to observe macroscopic evolution. The details about how these two are combined can be found in Subsection 3.2.

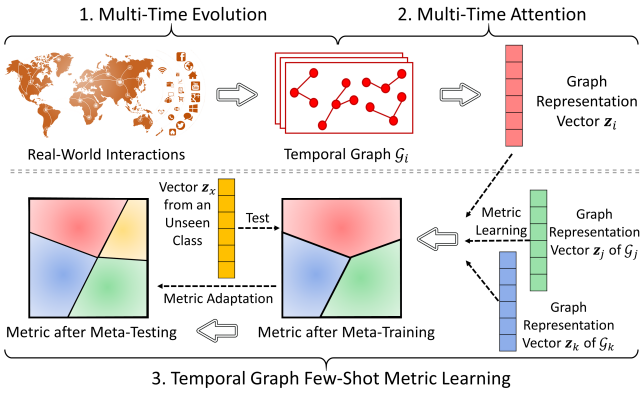


Figure 2: Overview of Temp-GFSM Framework.

For (C.4), after we provide exhaustive dynamics patterns for the learning process, we need to decide which patterns are dominating the similarity (or class labels) of temporal graphs that directly affects the utility of $\mathbb{M}$, e.g., the temporal graph classification accuracy. Therefore, we propose the *Multi-Time Attention* mechanism for assigning different attention weights to different evolution patterns. The details can be found in Subsection 3.3.

3 PROPOSED TEMP-GFSM FRAMEWORK

After discussing challenges (from C.1 to C.4) and proposing the corresponding solutions, we compose all these proposed techniques into an end-to-end trainable framework, named Temp-GFSM, which is responsible for learning an effective (e.g., accurate graph classification) and comprehensive (e.g., can be fast adapted to new unseen classes) metric via few-shot labeled temporal graph data. Before we dive into each module (from Subsection 3.2 to Subsection 3.4) of our Temp-GFSM, we would like to give an architectural overview of Temp-GFSM in Subsection 3.1. We use bold lowercase letters to denote column vectors (e.g., $\mathbf{a}$), bold capital letters to denote matrices (e.g., $\mathbf{A}$), and $\mathbf{A}(i,:)$ to denote the i-th row of $\mathbf{A}$. Also, we let the parenthesized superscript to denote the timestamp like $\mathbf{A}^{(t)}$.

3.1 Overview of Temp-GFSM

As shown in Figure 2, **first**, real-world interactions are modeled into a temporal graph $\mathcal{G}$ by *Multi-Time Evolution* such that the temporal graph $\mathcal{G}$ contains continuous time tracing the microscopic evolution patterns and discrete time tracing the macroscopic evolution patterns. **Second**, each temporal graph $\mathcal{G}$ goes through the *Multi-Time Attention* and gets represented by an embedding vector $\mathbf{z}$, which embeds different-level dynamics in $\mathcal{G}$ with different attention weights. **Third**, different class vectors go into the meta-training to seek for an effective metric space $\mathbb{M}$. After that, $\mathbb{M}$ gets updated when unseen class joins the meta-testing phase.

In Temp-GFSM, above three steps are combined by an end-to-end manner as follows. Given the multi-time evolution modeled temporal graphs and their labels $\mathcal{D} = \{(\mathcal{G}_1, y_1), \ldots, (\mathcal{G}_n, y_n)\}$, we split $\mathcal{D}$ into training set $\mathcal{D}^{train}$ for meta-training and $\mathcal{D}^{test}$ for meta-testing, where $\mathcal{D}^{test}$ only has unseen graph labels from $\mathcal{D}^{train}$.

- At the meta-training stage, we first shuffle $\mathcal{D}^{train}$ to sample *graph metric learning tasks* $\mathcal{T}_j$ following the distribution $\mathcal{T}_j \sim \mathbb{P}(\mathcal{T})$. Controlled by θ_j, each $\mathcal{T}_j$ is instanced by a *N-way K-shot temporal graph classification task* based on the graph representation $f_{\theta_j}(\mathcal{G}_i)$ (i.e., $\mathbf{z}_i$) in Euclidean space. To optimize θ_j, we again

sample a support set $\mathcal{D}^{train}_{support}$ and a query set $\mathcal{D}^{train}_{query}$ within each $\mathcal{T}_j$, such that $\mathcal{D}^{train}_{support}$ is used to train f_{θ_j} to accurately predict the labels of graphs from $\mathcal{D}^{train}_{query}$.
- After we transfer the learned knowledge (i.e., θ_j) from each observed task $\mathcal{T}_j$ to the meta-learner (i.e., Θ), then at the meta-testing stage, we fine-tune Θ a few times by classifying unseen temporal graphs only in $\mathcal{D}^{test}_{support}$. Finally, we report the classification accuracy of fine-tuned Θ in query set $\mathcal{D}^{test}_{query}$.

3.2 Multi-Time Evolution

Even in the same temporal graph, different modeling methods focus on different (partial) dynamic patterns. For example, a temporal graph can be modeled by an initial state G with a set of timestamped events O, and each event can be node/edge addition/deletion, which modeling is called continuous-time [30] or streaming [1]. Also, that temporal graph can be modeled as a sequence of time-respecting snapshots $G^{(1)}, G^{(2)}, \ldots$, and each $G^{(t)}$ has its own node set and edge set, which modeling is called discrete-time [30] or snapshot [1]. These two modeling methods have non-trivial complements [1, 30]: (1) **continuous-time** models rapid node/edge-level evolution, i.e., microscopic evolution [35], for example, protein molecule interactions in a cell [18]; (2) However, it loses control to represent the episodic and slowly-changing evolution patterns, which can be captured by **discrete-time modeling** (snapshot), i.e., macroscopic evolution [36], for example, the periodical metabolic cycles in that cell [18]. To the best of our knowledge, different evolution patterns in a single graph are currently not jointly considered for improving the graph representation learning.

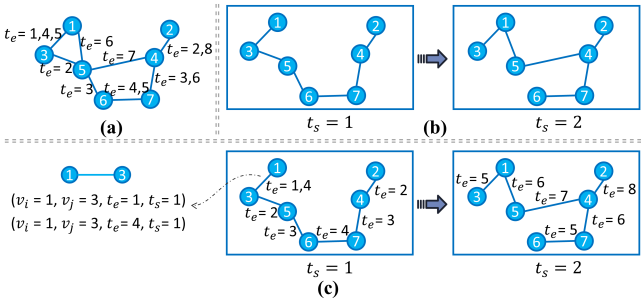


Figure 3: Part (a) shows a streaming graph with only edge timestamps t_e. Part (b) shows a snapshot-modeled graph with only snapshot timestamps t_s, where each t_s elapses every 4 t_e. Part (c) shows our multi-time evolution modeling with edge timestamps t_s and snapshot timestamps t_e.

To bridge this gap, in our multi-time evolution modeling, each temporal graph $\mathcal{G}$ is defined as $\mathcal{G} = (V, E, t_e, t_s)$. V stands for the set of nodes, where each one ever exists in the whole life time of $\mathcal{G}$. E is the set of temporal edges, where each edge is defined as $e = (v_i, v_j, t_e, t_s)$, where $t_e \in \{0, 1, \ldots, T_e\}$ is named as *edge timestamp* and $t_s \in \{0, 1, \ldots, T_s\}$ is named as *snapshot timestamp*. As shown in Figure 3, for the record $e = (v_i, v_j, t_e, t_s)$, at the edge level, (v_i, v_j, t_e) means the connection between v_i, v_j exists at time t_e; if it still exists in the future, t_e will increase correspondingly, and if it is deleted, then this tuple (v_i, v_j, t_e) does not exist, neither does (v_i, v_j, t_e, t_s). At the snapshot level, $e = (v_i, v_j, t_e, t_s)$ means that the event (v_i, v_j, t_e) happens in the interval of snapshot $\mathcal{S}^{(t_s)}$. According to [30], in our multi-time evolution modeling, the edge timestamp t_e is continuous, and snapshot timestamp t_s is discrete. Because each snapshot $\mathcal{S}^{(t_s)}$ at t_s is generated by aggregating the event at each t_e with a certain range (e.g., every 4 t_e in Figure 3).

Note that, two timestamps (t_e and t_s) are both real, and some temporal graphs have both naturally (furthermore, t_s do not need to be the equal interval in items of t_e like Figure 3). For example, in [18], each dynamic protein-protein interaction network (PPI) has 36 continuous observations (i.e., 36 edge timestamps), every 12 observations compose a metabolic cycle (i.e., 3 snapshot timestamps), and each cycle reflects 25 mins in the real world. Also, for those temporal graphs that only have edge timestamps, we can manually set the snapshot timestamp as a period hyperparameter (i.e., how many t_e compose a t_s) like Figure 3. Moreover, in experiments, we show how we set this hyperparameter to learn episodic or slowly-changed patterns to further help the learned metric utility (e.g., temporal graph classification). As for the data structure, we store each edge (v_i, v_j, t_e) in an array and each snapshot $\mathcal{S}^{(t_s)}$ by an adjacency matrix as $\mathbf{A}^{(t_s)} \in \mathbb{R}^{|V^{(t_s)}| \times |V^{(t_s)}|}$, i.e., $V^{(t_s)} \subseteq V$ and $|V^{(t_s)}| \neq |V^{(t_s+1)}|$ is allowable. For the notation clarity, we denote the node feature matrix $\mathbf{X} \in \mathbb{R}^{n \times m}$, such that the input node feature of temporal graph $\mathcal{G}$ is already time-aware, and $n = |V|$ and m denotes the dimension of features[1].

3.3 Multi-Time Attention

Given the multi-scope dynamics of temporal graph $\mathcal{G}$ modeled by Multi-Time Evolution, the next step in our Temp-GFSM is to assign different weights to different evolution patterns, such that the dominating evolution pattern(s) can be discovered to make the learned metric accurate for downstream tasks, e.g., temporal graph classification. Hence, we propose the *Multi-Time Attention* scheme. As shown in Figure 4, our proposed Multi-Time Attention has three sequential components, *Node-Level Lifelong Attention*, *Intra-Snapshot Attention*, and *Inter-Snapshot Attention*. Hence, a temporal graph $\mathcal{G}_i$ gets represented by an embedding vector $\mathbf{z}_i$ with its evolution patterns getting appropriate attention weights.

Node-Level Lifelong Attention. As shown in Figure 4, this mechanism first learns microscopic evolution patterns by the continuous timestamp t_e. To achieve this learning, we need to sample a time-aware adjacent node sequence $\mathcal{N}_v$ for each node v in the temporal graph $\mathcal{G}$ (as shown in Algorithm 1 in Appendix), then we apply the node-level time attention mechanism on $\mathcal{N}_v$ to learn the time-aware embedding $\mathbf{u}_v$ for each node v.

The self-attention mechanism has become the key component for representing sequential data [53], which itself could not deal with ordering information without a positional encoding function to map discrete position indexes into a differentiable functional domain. Therefore, we need a time encoding function $\mathbb{K}$ for our node-level time attention mechanism, which could map every observed time interval of node connections into a continuous differentiable functional domain, i.e., $\mathbb{K} : [t_e - \Delta t, t_e] \rightarrow \mathbb{R}^d$. Another intuition of involving $\mathbb{K}$ is that, suppose node v_i connects with node v_j at edge timestamp $t_e - \Delta t$, when we need to represent node v_j at edge timestamp t_e, we wish the encoded node representation $\mathbf{u}_{v_j}^{(t_e)}$ to be time-aware by containing the temporal relation information of v_i

[1] Our implementation is readily designed for evolving input features according to different timestamps, which means that $\mathbf{X}^{(t_s)} \in \mathbb{R}^{|V^{(t_s)}| \times m^{(t_s)}}$

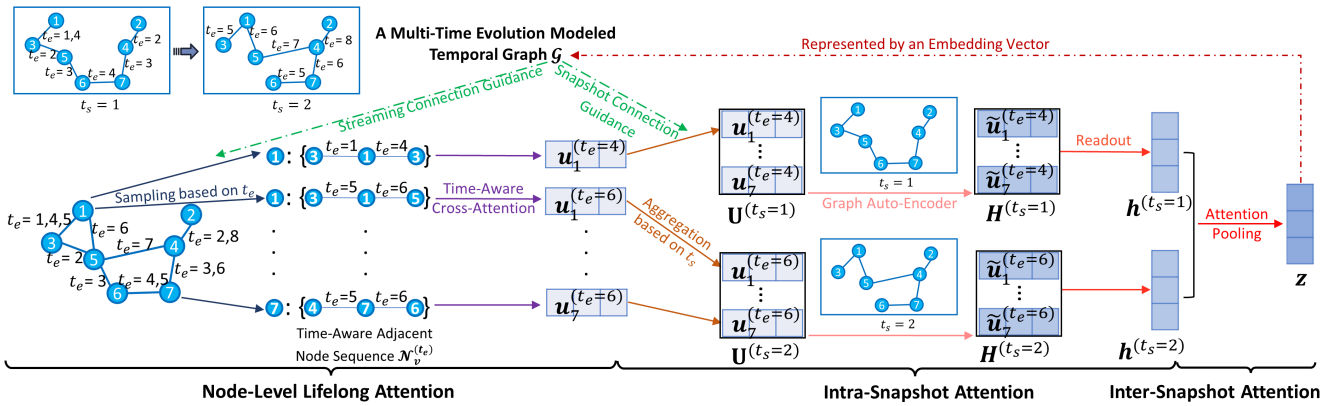


Figure 4: Multi-Time Attention Mechanism. A temporal graph $\mathcal{G}$ gets represented by the embedding vector z.

(i.e., $\mathbb{K}(t_e-\Delta t, t_e)$). This time function $\mathbb{K}$ should be able to reflect the time relation between $\mathbf{u}_{v_i}^{(t_e-\Delta t)}$ and $\mathbf{u}_{v_j}^{(t_e)}$, and many previous works solve $\mathbb{K}$ with different kinds of kernel methods [14, 60, 61, 69, 73]. For example, in [60, 61],

$$\mathbb{K}(t_e-\Delta t, t_e) = \Psi(t_e-(t_e-\Delta t)) = \Psi(\Delta t) \tag{2}$$

and

$$\Psi(\Delta t) = \sqrt{\frac{1}{d}}[\cos\omega_1(\Delta t), \cos\omega_2(\Delta t), \ldots, \cos\omega_d(\Delta t)] \tag{3}$$

where $\Delta t = t_e-(t_e-\Delta t)$ denotes the input time interval, and $\{\omega_1, \ldots, \omega_d\}$ are learnable parameters.

With the above time encoding (i.e., Eq. 3), we are now ready to learn node representation $\mathbf{u}_v^{(t_e)}$. Intuitively, we set node v as the query node to query and aggregate attention weights from nodes in $\mathcal{N}_v^{(t_e)}$. To be more specific, after sampling temporal adjacent nodes $\mathcal{N}_v^{(t_e)}$ for the target (or query) node v, we apply a self-attention like mechanism on $\mathcal{N}_v^{(t_e)}$ and then stack cross-attention layers to aggregate the target node and its neighbour nodes' hidden representations, to learn the time-aware node representation $\mathbf{u}_v^{(t_e)}$.

Similar with the self-attention mechanism [53], we first use the multi-head cross-attention module and form, for each head k, queries $\mathbf{Q}$, keys $\mathbf{K}$ and values $\mathbf{V}$. Then the time-aware node representation of each head k, $\mathbf{a}_v^{k(t_e)} \in \mathbb{R}^r$, can be computed as follows.

$$\mathbf{a}_v^{k(t_e)} = \text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{r}})\mathbf{V} \in \mathbb{R}^r \tag{4}$$

where $\mathbf{Q} = [\mathbf{X}(v,:)\|\mathbb{K}(t_e,t_e)]\cdot\mathbf{W}_Q$ is generated by the target node v. $\mathbf{K} = \mathbf{N}\cdot\mathbf{W}_K$ and $\mathbf{V} = \mathbf{N}\cdot\mathbf{W}_V$ are generated by the adjacent node sequence with number $|\mathcal{N}_v^{(t_e)}|$. $\mathbf{N} \in \mathbb{R}^{|\mathcal{N}_v^{(t_e)}|\times(m+d)}$ is the matrix whose rows are $[\mathbf{X}(v',:)\|\mathbb{K}(t,t_e)] \in \mathbb{R}^{(m+d)}$ from the adjacent node sequence $\mathcal{N}_v^{(t_e)}$, and $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{(m+d)\times r}$ are three learnable weight matrices with r denoting the dimension of the time-aware node representation vector $\mathbf{a}_v^{k(t_e)}$. We concatenate the output $\mathbf{a}_v^{k(t_e)}$ of each head k and feed it to a fully-connected feed-forward neural network, of which the parameters are $\mathbf{W}_0 \in \mathbb{R}^{k(m+d)\times r}$ and $\mathbf{b}_0 \in \mathbb{R}^r$, in order to generate the time-aware node representation $\mathbf{a}_v^{(t_e)}$. The aggregation of multi-head attention is as follows.

$$\mathbf{a}_v^{(t_e)} = ([\mathbf{a}_v^{1(t_e)}\|\ldots\|\mathbf{a}_v^{k(t_e)}])\mathbf{W}_0 + \mathbf{b}_0 \tag{5}$$

As mentioned above, one (multi-head) attention layer actually aggregates 1-hop neighbours temporal information according to the node adjacency sequence sampled by $\mathcal{N}_v^{(t_e)}$ Algorithm 1. Then, we propose to stack time-aware cross-attention layers to capture temporal information from multi-hop away neighbour nodes. Hence, in the l-th attention layer, the time-aware node representation $\mathbf{u}_v^{l(t_e)}$ can be computed as follows.

$$\mathbf{u}_v^{l(t_e)} = \text{ReLU}([\mathbf{a}_v^{l(t_e)}\|\mathbf{u}_v^{l-1(t_e)}]\mathbf{W}_1^l + \mathbf{b}_1^l)\mathbf{W}_2^l + \mathbf{b}_2^l \tag{6}$$

where $\mathbf{W}_1^l$, $\mathbf{b}_1^l$, $\mathbf{W}_2^l$, and $\mathbf{b}_2^l$ are parameters of the feed-forward neural network at layer l. Also, When $l=1$, $\mathbf{u}_v^{l-1(t_e)} = [\mathbf{X}(v,:)\|\mathbb{K}(t_e,t_e)]$.

Intra-Snapshot Attention. After we learn the time-aware node embedding $\mathbf{u}_v^{(t_e)}$ that follows the streaming pattern based on the edge timestamp t_e, we also want the node in the snapshot $\mathcal{S}^{(t_s)}$ follows the snapshot pattern of $\mathbf{A}^{(t_s)}$ w.r.t the snapshot timestamp t_s. Therefore, intra-snapshot time attention is proposed to add constraints on node embeddings $\mathbf{u}_v^{(t_e)}$ in terms of timestamp t_s by reconstructing $\mathbf{A}^{(t_s)}$ via a graph autoencoder.

Note that snapshot $\mathcal{S}^{(t_s)}$ may not have all nodes of the input temporal graph. Thus, we construct the snapshot feature matrix $\mathbf{U}^{(t_s)} \in \mathbb{R}^{|V^{(t_s)}|\times r}$ in which rows are time-aware node embedding vectors, i.e., for the edge (v_i, v_j, t_e, t_s), $\mathbf{U}^{(t_s)}(v_i,:) = \mathbf{u}_{v_i}^{(t_e)}$ and $\mathbf{U}^{(t_s)}(v_j,:) = \mathbf{u}_{v_j}^{(t_e)}$. If there is also another edge (v_i, v_j, t_e', t_s), we will sample the most recent edge timestamp. For example, if $t_e' > t_e$, then $\mathbf{U}^{(t_s)}(v_i,:) = \mathbf{u}_{v_i}^{(t_e')}$ and $\mathbf{U}^{(t_s)}(v_j,:) = \mathbf{u}_{v_j}^{(t_e')}$. As shown in Figure 4, for snapshot $\mathcal{S}^{(t_s=1)}$, there exists two edges connecting nodes v_i=1 and v_j=3, i.e., (v_i=1, v_j=3, t_e=1, t_s=1) and (v_i=1, v_j=3, t_e=4, t_s=1). Then, for node v_i=1, vector $\boldsymbol{u}_1^{(t_e=4)}$ other than $\boldsymbol{u}_1^{(t_e=1)}$ goes into $\boldsymbol{U}^{(t_s=1)}(1,:)$. The reason we adopt the latest node embedding is that, according to the sampling strategy in Algorithm 1, the latest node embedding contains early node information.

Now, with adjacency matrix $\mathbf{A}^{(t_s)}$ and snapshot feature matrix $\mathbf{U}^{(t_s)}$, we next learn the latent intra-snapshot representation matrix $\mathbf{H}^{(t_s)}$. Particularly, we add a reconstruction loss to refine the time-aware node embedding vectors in matrix $\mathbf{U}^{(t_s)}$ to learn $\mathbf{H}^{(t_s)}$ via the graph autoencoder model [31], and the snapshot reconstruction loss $\mathcal{L}_{rec}$ for the snapshot topology of $\mathcal{S}^{(t_s)}$ is defined as follows.

$$\mathcal{L}_{rec}(\mathbf{A}^{(t_s)}, \mathbf{U}^{(t_s)}) = \|\mathbf{A}^{(t_s)} - \hat{\mathbf{A}}^{(t_s)}\|_F \tag{7}$$

where $\hat{\mathbf{A}}^{(t_s)} = \sigma(\mathbf{H}^{(t_s)}\mathbf{H}^{(t_s)\top})$ is the reconstructed adjacency matrix computed by the sigmoid of the inner production of $\mathbf{H}^{(t_s)}$ and its transpose. $\mathbf{H}^{(t_s)} = \text{GNN}_{enc}(\mathbf{A}^{(t_s)}, \mathbf{U}^{(t_s)}) \in \mathbb{R}^{|V^{(t_s)}| \times q}$ denotes the intra-snapshot representation matrix, and $\|\cdot\|_F$ is the Frobenius norm. GNN_{enc} can be realized by GCN [32] or GAT [54]. Note that $\mathbf{H}^{(t_s)}$ and $\mathbf{U}^{(t_s)}$ share the same number of rows, but each column is updated from $\mathbf{u}_v^{(t_e)}$ to $\tilde{\mathbf{u}}_v^{(t_e)}$

Given the extracted intra-snapshot representation matrix $\mathbf{H}^{(t_s)}$, we apply a Readout function to get the intra-snapshot representation vector $\mathbf{h}^{(t_s)}$ for each snapshot timestmap t_s as follows.

$$\mathbf{h}^{(t_s)} = \text{Readout}(\mathbf{H}^{(t_s)}(v,:) \mid v \in \{1, \ldots, |V^{(t_s)}|\}) \in \mathbb{R}^q \tag{8}$$

where the Readout function could be a graph pooling layer like [66, 68] or any specifically designed attention pooling layers.

Inter-Snapshot Attention. After we obtain the intra-snapshot representation vector $\mathbf{h}^{(t_s)}$ for each snapshot timestamp t_s individually, we are not sure which one or ones should represent the temporal graph representation vector $\mathbf{z}$ to make it class-distinctive. For example, if a certain snapshot $\mathcal{S}$ is shared by different classes of temporal graphs, then that snapshot is less representative and we should decrease its weight during the snapshot representations aggregation process, to make different class temporal graph-level representation distinguishable. Therefore, we design a inter-snapshot time attention mechanism on vectors $\mathbf{h}^{(t_s)}$ to obtain the final temporal graph representation vector $\mathbf{z} \in \mathbb{R}^q$. Especially, the inter-snapshot time attention is realized by an attention pooling layer [3] with attention weight $\alpha^{(t_s)}$ for each snapshot time t_s. Moreover, that inter-snapshot time attention weight is parameterized by the learnable query vector $\mathbf{w}^{(t_s)} \in \mathbb{R}^q$. The temporal graph representation can then be calculated as follows.

$$\mathbf{z} = \sum_{t_s=1}^{T_s} (\alpha^{(t_s)} \mathbf{h}^{(t_s)}) \in \mathbb{R}^f, \ \alpha^{(t_s)} = \text{softmax}(\mathbf{w}^{(t_s)\top} \mathbf{h}^{(t_s)}) \tag{9}$$

3.4 Temporal Graph Few-Shot Metric Learning

For using few-shot labeled data to learn the metric, we involve the above discussed representation learning process into the few-shot learning setting, i.e., *Bi-Level Optimization*. First, in each graph metric learning task $\mathcal{T}_i$ during the meta-training, *Prototype Generator* generates the prototype for each class, such that each θ_i built on $\mathcal{D}^{train}_{support}$ can be in-depth optimized for the accurate classification on $\mathcal{D}^{train}_{query}$. Second, the meta-learner Θ transfers the knowledge from each θ_i of $\mathcal{T}_i$ in the meta-training phase to new tasks $\mathcal{T}_j$ in the meta-testing phase for the fast adaption on unseen classes.

Prototype Generator. After we encode a temporal graph $\mathcal{G}$ into a vector $\mathbf{z}$ as shown in Figure 1, we want the same class temporal graphs closer and different class graphs farther apart in metric $\mathbb{M}$, i.e., same class representation vectors are closer to their class prototype and farther from other class prototypes. Thus, in each graph metric learning task $\mathcal{T}_i$ with support set $\mathcal{D}^{train}_{support}$ and query set $\mathcal{D}^{train}_{query}$, we can first learn prototypes in $\mathcal{D}^{train}_{support}$ and then use them to predict graph class labels in $\mathcal{D}^{train}_{query}$. The construction of prototype $\mathbf{p}_k$ of class k in $\mathcal{D}^{train}_{support}$ is expressed as follows.

$$\mathbf{p}_k = \frac{1}{C_k} \sum_{j}^{C_k} (\mathbf{z}_j), \ \mathcal{G}_j \in \mathcal{D}^{train}_{support} \text{ and } y_j = k \tag{10}$$

where $\mathcal{G}_j$ is a graph with label y_j, $\mathbf{z}_j$ is the embedding of $\mathcal{G}_j$, and C_k denotes the number of temporal graphs in class k in $\mathcal{D}^{train}_{support}$.

To help the class prototype distinctive to each other, we design the temporal graph classification loss $\mathcal{L}_{cls}$ in each graph metric learning task $\mathcal{T}_i$ to tune θ_i in $\mathcal{D}^{train}_{query}$.

$$\mathcal{L}_{cls} = -\sum_{j}^{C_k} \log \frac{\exp(-dist(\mathbf{z}_j, \mathbf{p}_k))}{\sum_{\bar{k}} \exp(-dist(\mathbf{z}_j, \mathbf{p}_{\bar{k}}))}, \quad \mathcal{G}_j \in \mathcal{D}^{train}_{query} \text{ and } y_j = k \tag{11}$$

where $\mathbf{p}_k$ denotes the prototype of class k learned from $\mathcal{D}^{train}_{support}$, $\bar{k}$ denote the class other than k, $dist(\cdot)$ denotes Euclidean distance between two vectors, $\mathbf{z}_j$ is the representation vector of $\mathcal{G}_j$, and C_k denotes the number of class k graphs in $\mathcal{D}^{train}_{query}$.

Bi-Level Optimization. We have introduced the whole learning procedure with two loss functions $\mathcal{L}_{rec}$ (i.e., Eq. 7) and $\mathcal{L}_{cls}$ (i.e., Eq. 11). The entire loss is the weighted sum (i.e., by the balancing hyperparameter γ, $\mathcal{L}_{cls} + \gamma\mathcal{L}_{rec}$) for extracting knowledge θ_i from a single task $\mathcal{T}_i$. Next, we need to break though the knowledge transfer and adaption cross tasks given only few-shot examples. Here, we introduce a meta-learner to transfer the learned knowledge θ_i and tailor the globally shared knowledge Θ, the theory behind is that transferring shareable knowledge could obtain the fast convergence on unseen tasks [9, 38], and the bi-level optimization is able to find meta-learner Θ that could be fast converged in each graph metric learning task [16].

The detailed meta-training process is shown in Algorithm 2 in Appendix, and the detailed meta-testing process of Temp-GFSM is similar to Algorithm 2, i.e., after changing $\mathcal{D}^{train}_{support}$ into $\mathcal{D}^{test}_{support}$ and $\mathcal{D}^{train}_{query}$ into $\mathcal{D}^{test}_{query}$, the only difference is changing Step 10 of Algorithm 2 to directly report the accuracy based on θ_i instead of updating a new Θ.

4 EXPERIMENTS

In this section, we test Temp-GFSM in terms of the temporal graph classification accuracy and convergence speed. More experimental details like data preprocessing (e.g., how to set t_e and t_s if anyone is not given), parameter sensitivity analysis, and ablation study about multiple dynamics can be found in Appendix.

4.1 Experiment Setup

Datasets. Our experiments include 12 temporal graph datasets (i.e., 12 classes) from the biological domain [18] and 6 temporal graph datasets (i.e., 12 classes) from the social network domain [39]. Each biological graph is a dynamic protein-protein interaction network, where each edge describes the protein-protein interaction of metabolic cycles of yeast cells. Each social graph is a human-contact relation network, where the edges between two individuals stand for the online or offline contacts. The statistics of all network data are summarized in Table 1 and Table 2.

Baselines. The selection of baseline algorithms includes two aspects, i.e., *graph kernel or graph metric learning* and *static or dynamic*. Graph kernel methods include Shortest Path [7], Neighborhood Hash [26], and Weisfeiler-Lehman Optimal Assignment [33]. Graph metric learning or graph representation learning algorithms include GL2Vec [10], Graph2Vec [40], tdGraphEmbed [6], TGAT [61], and

Table 1: Statistics of Biological Temporal Graph Data

Graph	#Classes	#Graphs	Total Nodes	Total Edges	Timestamps	Graph	#Classes	#Graphs	Total Nodes	Total Edges	Timestamps
Uetz	1	11	922	2,159	36	Ito	1	11	2,856	8,638	36
Ho	1	11	1,548	42,220	36	Gavin	1	11	2,541	140,040	36
Krogan-LCMS	1	11	2,211	85,133	36	Krogan-MALDI	1	11	2,099	78,297	36
Yu	1	11	1,163	3,602	36	Breitkreutz	1	11	869	39,250	36
Babu	1	11	5,003	111,466	36	Lambert	1	11	697	6,654	36
Tarassov	1	11	1,053	4,826	36	Hazbun	1	11	143	1,959	36

Table 2: Statistics of Social Temporal Graph Data

Graph (Online)	#Classes	#Graphs	Total Nodes	Total Edges	Timestamps	Graph (Offline)	#Classes	#Graphs	Total Nodes	Total Edges	Timestamps
Facebook	2	995	95,224	267,673	104	Infectious	2	200	10,000	91,944	48
Tumblr	2	373	19,811	74,520	89	HighSchool	2	180	9,418	98,066	203
DBLP	2	755	39,917	241,674	46	MIT	2	97	1,940	142,508	5,576

Table 3: Temporal Graph Classification Accuracy on Biological Temporal Graphs

Method \ Few-shot Setting		3 way - 5 shot	3 way - 3 shot	3 way - 2 shot	3 way - 1 shot
Graph Kernel	Weisfeiler-Lehman Opt + kNN	0.6833 ± 0.1486	0.5722 ± 0.1554	0.4958 ± 0.1843	0.3417 ± 0.1371
	Neighborhood Hash + kNN	0.6833 ± 0.1858	0.5972 ± 0.1222	0.5833 ± 0.2050	0.5500 ± 0.2082
	Shortest Path + kNN	0.5433 ± 0.1988	0.5306 ± 0.0728	0.5292 ± 0.0946	0.2333 ± 0.1217
Graph Metric Learning	GL2Vec + kNN	0.0717 ± 0.0900	0.0917 ± 0.0793	0.0333 ± 0.0471	0.0333 ± 0.0667
	Graph2Vec + kNN [2]	–	–	–	–
	TGAT + kNN	0.1200 ± 0.0960	0.1250 ± 0.0793	0.1208 ± 0.0865	0.0917 ± 0.0319
	CAW + kNN	0.0400 ± 0.0362	0.0435 ± 0.0441	0.0569 ± 0.0370	0.0528 ± 0.0210
	tdGraphEmbed + kNN	0.2167 ± 0.1736	0.1056 ± 0.0814	0.1500 ± 0.1800	0.0750 ± 0.0877
	GL2Vec + Protonet	0.7100 ± 0.0361	0.6625 ± 0.0407	0.6075 ± 0.0496	0.5750 ± 0.0537
	Graph2Vec + ProtoNet	0.3792 ± 0.0459	0.3958 ± 0.0731	0.3958 ± 0.0241	0.3958 ± 0.0798
	TGAT + ProtoNet	0.2417 ± 0.0500	0.3083 ± 0.0739	0.2917 ± 0.1167	0.2417 ± 0.0319
	CAW + ProtoNet	0.1496 ± 0.0104	0.2113 ± 0.0110	0.2404 ± 0.0117	0.2842 ± 0.0044
	tdGraphEmbed + ProtoNet	0.6562 ± 0.1882	0.6791 ± 0.1141	0.6271 ± 0.1159	0.4229 ± 0.0463
	Temp-GFSM (Ours)	**0.7292 ± 0.0682**	**0.7917 ± 0.1278**	**0.7062 ± 0.0762**	**0.6833 ± 0.0589**

Table 4: Temporal Graph Classification Accuracy on Social Temporal Graphs

Method \ Few-shot Setting		3 way - 5 shot	3 way - 3 shot	3 way - 2 shot	3 way - 1 shot
Graph Kernel	Weisfeiler-Lehman Opt + kNN	0.3631 ± 0.0298	0.2941 ± 0.1023	0.2700 ± 0.1105	0.2133 ± 0.0388
	Neighborhood Hash + kNN	0.3938 ± 0.0371	0.3185 ± 0.1030	0.2178 ± 0.1500	0.3022 ± 0.1137
	Shortest Path + kNN	0.3996 ± 0.0317	0.3296 ± 0.1413	0.3556 ± 0.1139	0.3844 ± 0.0420
Graph Metric Learning	GL2Vec + kNN	0.2716 ± 0.0839	0.2637 ± 0.0482	0.1711 ± 0.0700	0.0000 ± 0.0000
	Graph2Vec + kNN	0.3360 ± 0.0352	0.3756 ± 0.0241	0.2400 ± 0.0149	0.1933 ± 0.0723
	TGAT + kNN	0.0289 ± 0.0096	0.0407 ± 0.0123	0.0333 ± 0.0068	0.0200 ± 0.0199
	CAW + kNN	0.0284 ± 0.0106	0.0378 ± 0.0178	0.0322 ± 0.0099	0.0333 ± 0.0192
	tdGraphEmbed + kNN	0.3600 ± 0.0208	0.3000 ± 0.1310	0.2767 ± 0.1185	0.2267 ± 0.0268
	GL2Vec + Protonet	0.3400 ± 0.0306	0.3822 ± 0.0290	0.2633 ± 0.0606	0.1933 ± 0.0723
	Graph2Vec + Protonet	0.3573 ± 0.0203	0.3711 ± 0.0279	0.2467 ± 0.0380	0.1933 ± 0.0723
	TGAT + ProtoNet	0.3227 ± 0.0171	0.3293 ± 0.0156	0.3243 ± 0.0110	0.3363 ± 0.0010
	CAW + ProtoNet	0.3340 ± 0.0113	0.3333 ± 0.0229	0.3380 ± 0.0155	0.3270 ± 0.0189
	tdGraphEmbed + ProtoNet	0.5083 ± 0.0121	0.4523 ± 0.0353	0.4670 ± 0.0199	0.3973 ± 0.0164
	Temp-GFSM (Ours)	**0.6161 ± 0.0139**	**0.5931 ± 0.0148**	**0.6074 ± 0.0164**	**0.5605 ± 0.0201**

CAW [59]. Among those, tdGraphEmbed is a dynamic algorithm that could take a temporal graph as input and output the graph embedding of each snapshot individually, and TGAT and CAW are dynamic graph representation learning algorithms but focus on the node-level representation learning. To enable graph kernel and metric learning methods the few-shot learning capability, we also include ProtoNet and its special case kNN method [50]. To enable static baselines to handle evolving graphs, we use Reduced Graph Representation [41] to map temporal graphs into dynamics-preserving static graphs. We use the mean pooling function for TGAT, CAW, and tdGraphEmbed when an aggregation is necessary.

[2]Cannot get the results within 48 hours.

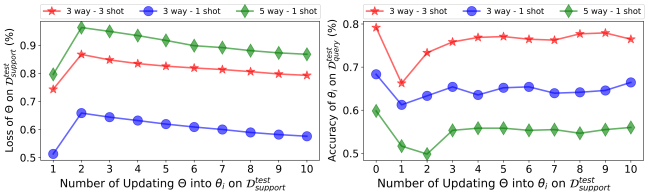
4.2 Temporal Graph Classification

First, in the biological dataset, given the 12 classes, we divide 8 classes into meta-training set $\mathcal{D}^{train}$ and 4 classes into meta-testing set $\mathcal{D}^{test}$. Note that $\mathcal{D}^{train}$ and $\mathcal{D}^{test}$ do not share any class label. We shuffle $\mathcal{D}^{train}$ and $\mathcal{D}^{test}$ 4 times for the cross-validation and report the average classification accuracy in Table 3, where our Temp-GFSM outperforms all baseline algorithms. For example, in the 3-way 3-shot setting, our Temp-GFSM achieves 79.17% temporal graph classification accuracy, which is 16.58% higher than the second place. An intuitive explanation is that compared with single-dynamics baselines (e.g., TGAT and CAW for streaming, and tdGraphEmbed for snapshots), Temp-GFSM could encode multiple

evolution patterns of protein networks and weigh them appropriately. Second, in the social dataset, the amount of offline graph data is less than online data as shown in Table 2. Thus, we aim to investigate whether sufficient online temporal graph data could provide transferable knowledge to help the offline temporal graph classification. To this end, we select online networks (i.e., Facebook, Tumblr, and DBLP) for $\mathcal{D}^{train}$ and offline networks (i.e., Infectious, HighSchool, and MIT) for $\mathcal{D}^{test}$. By comparing the performance in Table 4, we know that our Temp-GFSM achieves the best performance. An intuitive explanation is that online network patterns indeed share some knowledge that helps identify offline patterns. Also, our experiments show that increasing the number of shots during the meta-training is not always the good choice for improving the performance of meta-testing [3] because intra-class variances may be amplified [8].

4.3 Convergence during Meta-Testing

Following the same setting on the biological temporal graph data, we vary the number of shots and the number of ways during the meta-training [4], to investigate the convergence of our Temp-GFSM in the meta-testing phase. We report the loss of Θ on $\mathcal{D}^{test}_{support}$ and the accuracy of θ_i on $\mathcal{D}^{test}_{query}$ by every update of Θ on $\mathcal{D}^{test}_{support}$. As shown in Figure 5, we observe that our Temp-GFSM could achieve the fast convergence to the unseen tasks with only a few updates (e.g. 3 or 4 times) of the meta-learner like [16]. Interestingly, the ideal case occurs that the meta-training and meta-testing tasks share the global knowledge, i.e., Θ could perform well even without parameters update on $\mathcal{D}^{test}_{support}$, which partially prove that the prototype construction could deal with the zero-shot learning [50].



(a) Loss of Θ on $\mathcal{D}^{test}_{support}$ before Different Update Iterations.

(b) Accuracy of θ_i on $\mathcal{D}^{test}_{query}$ after Different Iterations.

Figure 5: Convergence Speed of Temp-GFSM on Bio Data.

5 RELATED WORK

Graph Metric Learning. Learning a good metric in the input feature space can be transferred to learn proper graph representation in Euclidean space, then graph embedding based graph metric learning methods are proposed [4, 37, 47]. Facing the label scarcity problem, many generic metric learning methods consider the few-shot learning or meta-learning strategy [2, 42, 50] to adapt knowledge across different tasks [74, 75] with only a few labeled samples in each task. Inspired by that, several graph metric learning methods cooperate with the few-shot learning manner, but most of these algorithms focus on learning the node-level metric across different graphs [13, 29, 34, 51, 58, 64]. Only a few graph metric learning methods learn the metric for the whole graph level to distinguish the similarity [9, 38]. Currently, graph few-shot metric learning methods ignore considering the dynamics of graphs into the metric learning process. We are the first effort to involve temporal dependencies of entire graphs into the learned metric via the few-shot learning. **Graph Kernel**. Given a distance metric $\mathbb{M}$, it should maintain four properties: non-negativity (i.e., $\mathbb{M}(\mathbf{x}, \mathbf{y}) \geq 0$), coincidence (i.e., $\mathbb{M}(\mathbf{x}, \mathbf{y}) = 0$ iff $\mathbf{x} = \mathbf{y}$), symmetry (i.e., $\mathbb{M}(\mathbf{x}, \mathbf{y}) = \mathbb{M}(\mathbf{y}, \mathbf{x})$), and subadditivity (i.e., $\mathbb{M}(\mathbf{x}, \mathbf{y}) + \mathbb{M}(\mathbf{y}, \mathbf{z}) \geq \mathbb{M}(\mathbf{x}, \mathbf{z})$) [57]. While in the graph kernel research, only the symmetry and non-negativity need to be satisfied for a kernel function, i.e., the symmetric graph kernel function $\mathbb{K}$ should be a positive semi-definite function [56]. To measure the similarity among graphs, one category graph kernel methods explicitly define the kernel function from the graph topological view, such as Random Walk graph kernel [56] and Weisfeiler-Lehman graph kernel [48]. To handle the evolving graph scenario, some methods map dynamic graphs into constant representation and then apply static graph kernel functions for dynamic node classification [65] and temporal graph classification [41]. On the other hand, another category graph kernel methods learn the kernel function instead of hand-crafted designing it [63, 70]. For example, in [63], the kernel is determined by learning the latent representation of substructures of input graphs.

6 CONCLUSION

In this paper, we propose a temporal graph few-shot metric learning framework, named Temp-GFSM. In Temp-GFSM, firstly, multi-time evolution modeling describes multiple dynamics of a single temporal graph. Secondly, the multi-time attention scheme is proposed to weigh and aggregate each evolution pattern to optimize the metric utility. Third, we involve the bi-level optimization in Temp-GFSM to achieve the fast adaption of the learned metric to unseen classes. We execute extensive experiments to show the effectiveness of our Temp-GFSM with different categorical baselines.

ACKNOWLEDGEMENT

This work is supported by the National Science Foundation (Award Number IIS-1947203, IIS-2117902, and IIS-2137468), the U.S. Department of Homeland Security (Award Number 17STQAC00001-05-00), and in part by the National Institute on Aging of the NIH (Award Number P01AG039347). The views and conclusions are those of the authors and should not be interpreted as representing the official policies of the funding agencies or the government.

REFERENCES

[1] Charu C. Aggarwal and Karthik Subbian. Evolutionary network analysis: A survey. *ACM Comput. Surv.*, 2014.
[2] Kelsey R. Allen, Evan Shelhamer, Hanul Shin, and Joshua B. Tenenbaum. Infinite mixture prototypes for few-shot learning. In *ICML*, 2019.
[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.
[4] Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. Simgnn: A neural network approach to fast graph similarity computation. In *WSDM*, 2019.
[5] Aharon Bar-Hillel, Tomer Hertz, Noam Shental, and Daphna Weinshall. Learning a mahalanobis metric from equivalence constraints. *J. Mach. Learn. Res.*, 2005.
[6] Moran Beladev, Lior Rokach, Gilad Katz, Ido Guy, and Kira Radinsky. tdgraphembed: Temporal dynamic graph-level embedding. In *CIKM*, 2020.
[7] Karsten M. Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *ICDM*, 2005.
[8] Tianshi Cao, Marc T. Law, and Sanja Fidler. A theoretical analysis of the number of shots in few-shot learning. In *ICLR*, 2020.
[9] Jatin Chauhan, Deepak Nathani, and Manohar Kaul. Few-shot learning on graphs via super-classes based on graph spectral measures. In *ICLR*, 2020.

[3] The number of shots in the query set during meta-testing is 2.
[4] We randomly pick one class from $\mathcal{D}^{train}$ and add it to $\mathcal{D}^{test}$ for the 5-way 1-shot setting.

[10] Hong Chen and Hisashi Koga. Gl2vec: Graph embedding enriched by line graphs with edge features. In *ICONIP*, 2019.
[11] Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *ICML*, 2016.
[12] Tyler Derr, Yao Ma, Wenqi Fan, Xiaorui Liu, Charu C. Aggarwal, and Jiliang Tang. Epidemic graph convolutional network. In *WSDM*, 2020.
[13] Kaize Ding, Jianling Wang, Jundong Li, Kai Shu, Chenghao Liu, and Huan Liu. Graph prototypical networks for few-shot learning on attributed networks. In *CIKM*, 2020.
[14] Nan Du, Hanjun Dai, Rakshit Trivedi, Utkarsh Upadhyay, Manuel Gomez-Rodriguez, and Le Song. Recurrent marked temporal point processes: Embedding event history to vector. In *KDD*, 2016.
[15] David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *NeurIPS*, 2015.
[16] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017.
[17] Dongqi Fu, Yikun Ban, Hanghang Tong, Ross Maciejewski, and Jingrui He. DISCO: comprehensive and explainable disinformation detection. *CoRR*, 2022.
[18] Dongqi Fu and Jingrui He. DPPIN: A biological repository of dynamic protein-protein interaction network data. *CoRR*, 2021.
[19] Dongqi Fu and Jingrui He. SDG: A simplified and dynamic graph neural network. In *SIGIR*, 2021.
[20] Dongqi Fu, Zhe Xu, Bo Li, Hanghang Tong, and Jingrui He. A view-adversarial framework for multi-view network embedding. In *CIKM*, 2020.
[21] Dongqi Fu, Dawei Zhou, and Jingrui He. Local motif clustering on time-evolving graphs. In *KDD*, 2020.
[22] Zoubin Ghahramani. Bayesian non-parametrics and the probabilistic approach to modelling. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, (1984), 2013.
[23] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017.
[24] Amir Globerson and Sam T. Roweis. Metric learning by collapsing classes. In *NeurIPS*, 2005.
[25] Jacob Goldberger, Sam T. Roweis, Geoffrey E. Hinton, and Ruslan Salakhutdinov. Neighbourhood components analysis. In *NeurIPS*, 2004.
[26] Shohei Hido and Hisashi Kashima. A linear-time graph kernel. In *ICDM*, pages 179–188, 2009.
[27] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay S. Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *ICLR*, 2020.
[28] Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. GPT-GNN: generative pre-training of graph neural networks. In *KDD*, 2020.
[29] Kexin Huang and Marinka Zitnik. Graph meta learning via local subgraphs. In *NeurIPS*, 2020.
[30] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Representation learning for dynamic graphs: A survey. *JMLR*, 2020.
[31] Thomas N. Kipf and Max Welling. Variational graph auto-encoders. *CoRR*, 2016.
[32] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.
[33] Nils M. Kriege, Pierre-Louis Giscard, and Richard C. Wilson. On valid optimal assignment kernels and applications to graph classification. In *NeurIPS*, 2016.
[34] Lin Lan, Pinghui Wang, Xuefeng Du, Kaikai Song, Jing Tao, and Xiaohong Guan. Node classification on graphs with few-shot novel labels via meta transformed network embedding. In *NeurIPS*, 2020.
[35] Jure Leskovec, Lars Backstrom, Ravi Kumar, and Andrew Tomkins. Microscopic evolution of social networks. In *KDD*, 2008.
[36] Jure Leskovec, Jon M. Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *KDD*, 2005.
[37] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects. In *ICML*, 2019.
[38] Ning Ma, Jiajun Bu, Jieyu Yang, Zhen Zhang, Chengwei Yao, Zhi Yu, Sheng Zhou, and Xifeng Yan. Adaptive-step graph meta-learner for few-shot graph classification. In *CIKM*, 2020.
[39] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *CoRR*, 2020.
[40] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. graph2vec: Learning distributed representations of graphs. *CoRR*, 2017.
[41] Lutz Oettershagen, Nils M. Kriege, Christopher Morris, and Petra Mutzel. Temporal graph kernels for classifying dissemination processes. In *SDM*, 2020.
[42] Boris N. Oreshkin, Pau Rodríguez López, and Alexandre Lacoste. TADAM: task dependent adaptive metric for improved few-shot learning. In *NeurIPS*, 2018.
[43] Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. GCC: graph contrastive coding for graph neural network pre-training. In *KDD*, 2020.
[44] Benedek Rozemberczki, Oliver Kiss, and Rik Sarkar. Karate club: An API oriented open-source python framework for unsupervised learning on graphs. In *CIKM*, 2020.
[45] Ruslan Salakhutdinov and Geoffrey E. Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. In *AISTATS*, 2007.
[46] Bernhard Schölkopf, Koji Tsuda, and Jean-Philippe Vert. *Kernel Methods in Computational Biology*. MIT press, 2004.
[47] Blake Shaw, Bert Huang, and Tony Jebara. Learning a distance metric from a network. In *NeurIPS*, 2011.
[48] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.*, 2011.
[49] Giannis Siglidis, Giannis Nikolentzos, Stratis Limnios, Christos Giatsidis, Konstantinos Skianis, and Michalis Vazirgiannis. Grakel: A graph kernel library in python. *J. Mach. Learn. Res.*, 2020.
[50] Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical networks for few-shot learning. In *NeurIPS*, 2017.
[51] Qiuling Suo, Jingyuan Chou, Weida Zhong, and Aidong Zhang. Tadanet: Task-adaptive network for graph-enriched meta-learning. In *KDD*, 2020.
[52] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, Alexander M. Bronstein, and Emmanuel Müller. Netlsd: Hearing the shape of a graph. In *KDD*, 2018.
[53] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.
[54] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.
[55] Oriol Vinyals, Charles Blundell, Tim Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In *NeurIPS*, 2016.
[56] S. V. N. Vishwanathan, Nicol N. Schraudolph, Risi Kondor, and Karsten M. Borgwardt. Graph kernels. *J. Mach. Learn. Res.*, 2010.
[57] Fei Wang and Jimeng Sun. Survey on distance metric learning and dimensionality reduction in data mining. *Data Min. Knowl. Discov.*, 2015.
[58] Ning Wang, Minnan Luo, Kaize Ding, Lingling Zhang, Jundong Li, and Qinghua Zheng. Graph few-shot learning with attribute matching. In *CIKM*, 2020.
[59] Yanbang Wang, Yen-Yu Chang, Yunyu Liu, Jure Leskovec, and Pan Li. Inductive representation learning in temporal networks via causal anonymous walks. In *ICLR*, 2021.
[60] Da Xu, Chuanwei Ruan, Evren Körpeoglu, Sushant Kumar, and Kannan Achan. Self-attention with functional time representation learning. In *NeurIPS*, 2019.
[61] Da Xu, Chuanwei Ruan, Evren Körpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. In *ICLR*, 2020.
[62] Yuchen Yan, Lihui Liu, Yikun Ban, Baoyu Jing, and Hanghang Tong. Dynamic knowledge graph alignment. In *AAAI*, 2021.
[63] Pinar Yanardag and S. V. N. Vishwanathan. Deep graph kernels. In *KDD*, 2015.
[64] Huaxiu Yao, Chuxu Zhang, Ying Wei, Meng Jiang, Suhang Wang, Junzhou Huang, Nitesh V. Chawla, and Zhenhui Li. Graph few-shot learning via knowledge transfer. In *AAAI*, 2020.
[65] Yibo Yao and Lawrence B. Holder. Scalable svm-based classification in dynamic graphs. In *ICDM*, 2014.
[66] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*, 2018.
[67] Tomoki Yoshida, Ichiro Takeuchi, and Masayuki Karasuyama. Learning interpretable metric between graphs: Convex formulation and computation with graph mining. In *KDD*, 2019.
[68] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *AAAI*, 2018.
[69] Qiang Zhang, Aldo Lipani, Ömer Kirnap, and Emine Yilmaz. Self-attentive hawkes process. In *ICML*, 2020.
[70] Qi Zhao and Yusu Wang. Learning metrics for persistence-based summaries and applications for graph classification. In *NeurIPS*, 2019.
[71] Dawei Zhou, Lecheng Zheng, Jiawei Han, and Jingrui He. A data-driven graph generative model for temporal interaction networks. In *KDD*, 2020.
[72] Dawei Zhou, Lecheng Zheng, Jiejun Xu, and Jingrui He. Misc-gan: A multi-scale generative model for graphs. *Frontiers Big Data*, 2019.
[73] Ke Zhou, Hongyuan Zha, and Le Song. Learning triggering kernels for multi-dimensional hawkes processes. In *ICML*, 2013.
[74] Yao Zhou, Lei Ying, and Jingrui He. Multic^2: an optimization framework for learning from task and worker dual heterogeneity. In *SDM*, 2017.
[75] Yao Zhou, Lei Ying, and Jingrui He. Multi-task crowdsourcing via an optimization framework. *ACM Trans. Knowl. Discov. Data*, 2019.
[76] Marinka Zitnik, Rok Sosič, and Jure Leskovec. Prioritizing network communities. *Nature communications*, 2018.

A NOTATION

Table 5: Table of Notation

Symbol	Definition and Description
$\mathcal{D}$	temporal graph set $\{(\mathcal{G}_0, y_0), (\mathcal{G}_1, y_1), \ldots, (\mathcal{G}_n, y_n)\}$
$\mathcal{G}_j$	the j-th temporal graph with snapshots $\{\mathcal{S}_j^{(t_s)}\}_{t_s=1}^{T_s}$
$\mathcal{S}_j^{(t_s)}$	the t_s timestamped snapshot of the j-th temporal graph $\mathcal{G}_j$
t_e, t_s	edge timestamp, snapshot timestamp
m	input node feature dimension
d	time feature dimension
r	node-level embedding dimension
q	snapshot-level and graph-level embedding dimension
$\mathcal{T}_i$	the i-th graph metric learning task sampled from $\mathbb{P}(\mathcal{T})$
θ_i	all learnable parameters of task $\mathcal{T}_i$
Θ	meta-learner over all tasks in meta-training
$\mathcal{D}_{support}^{train}$	set of support temporal graphs in meta-training
$\mathcal{D}_{query}^{train}$	set of query temporal graphs in meta-training
$\mathcal{D}_{support}^{test}$	set of support temporal graphs in meta-testing
$\mathcal{D}_{query}^{test}$	set of query temporal graphs in meta-testing

B ALGORITHM

The time-aware adjacent node sampling is summarized in Algorithm 1. Starting from a user-initialized timestamp and a target node, Algorithm 1 captures its time-aware 1-hop neighbours, which actually forms a star network surrounding the target node at a specific temporal scope. Note that, if a node v' connects the target node v at different timestamps, then each v' is a different node towards the target node v.

Algorithm 1 Sample Time-Aware Adjacent Node Sequence $\mathcal{N}_v^{(t_e)}$ for Node v at Edge Timestamp t_e

Input: target node v at t_e, temporal graph $\mathcal{G}$
Output:
time-aware adjacent node sequence $\mathcal{N}_v^{(t_e)}$

Initialize the starting timestamp t
while $t \leq t_e$ **do**
  if edge (v', v, t) exists **then** ▷ existing connections before t_e
    $\mathcal{N}_v^{(t_e)}$ appends $\mathbf{X}(v', :) \| \mathbb{K}(t, t_e)$ ▷ concatenation of $\mathbf{X}(v', :) \in \mathbb{R}^m$ and $\mathbb{K}(t, t_e) \in \mathbb{R}^d$
  end if
  t++
end while

The meta-training phase is shown in Algorithm 2, we randomly initialize Θ in Step 1. Then, in each graph metric learning task $\mathcal{T}_i \sim \mathbb{P}(\mathcal{T})$, we obtain the temporal graph representation vector in Step 5 and build the class prototype for each class of the support set in Step 6. In Step 8, we tune Θ to get θ_i for current task $\mathcal{T}_i$. In Step 10, we aggregate the loss from each task $\mathcal{T}_i$ and fine tune Θ to end the meta-training process. After that, we can use the fine-tuned Θ as the initialized parameter in the meta-testing stage for unseen graph metric learning tasks, aiming to the fast adaptation via only a few labeled samples. The meta-testing phase of Temp-GFSM is similar to Algorithm 2. After changing $\mathcal{D}_{support}^{train}$ into $\mathcal{D}_{support}^{test}$ and $\mathcal{D}_{query}^{train}$ into $\mathcal{D}_{query}^{test}$, the only difference is that Step 10 directly reports the accuracy based on θ_i instead of getting new Θ.

Algorithm 2 Meta-Training Process of Temp-GFSM

Input: graph metric learning task distribution $\mathbb{P}(\mathcal{T})$, step size hyperparameters α and β, loss balancing hyperparameter γ

1: Randomly initialize Θ ▷ Θ denotes all parameters
2: **while** not done **do**
3: Sample task $\mathcal{T}_i \sim \mathbb{P}(\mathcal{T})$ with its support set $\mathcal{D}_{support}^{train}$ and query set $\mathcal{D}_{query}^{train}$
4: **for** support set of each task $\mathcal{T}_i$ **do**
5: Compute $\mathbf{z}_j = f_\Theta(\mathcal{G}_j)$ ▷ $\mathcal{G}_j \in \mathcal{D}_{support}^{train}$
6: Construct $\mathbf{p}_k$ for each class k in $\mathcal{D}_{support}^{train}$ according to Eq. 10.
7: Evaluate snapshot reconstruction loss $\nabla_\Theta \mathcal{L}_{rec,\mathcal{T}_i}(f_\Theta)$ and classification loss $\nabla_\Theta \mathcal{L}_{cls,\mathcal{T}_i}(f_\Theta)$
8: Compute parameter $\theta_i \leftarrow \Theta - \alpha(\nabla_\Theta \mathcal{L}_{cls,\mathcal{T}_i}(f_\Theta) + \gamma \nabla_\Theta \mathcal{L}_{rec,\mathcal{T}_i}(f_\Theta))$
9: **end for**
10: Update $\Theta \leftarrow \Theta - \beta \nabla_\Theta \sum_{\mathcal{T}_i} (\mathcal{L}_{cls,\mathcal{T}_i}(f_{\theta_i}) + \gamma \mathcal{L}_{rec,\mathcal{T}_i}(f_{\theta_i}))$ ▷ On query set $\mathcal{D}_{query}^{train}$
11: **end while**

C PREPROCESSING OF BIOLOGICAL DATA

As shown in Table 1, there are 36 timestamps in each class, and we adopt them as edge timestamps t_e. According to [18], 11 graphs of each class in Table 1 are formed in this way: $t_e = \{1, 2, 3, 4, 5\}$ composes the first temporal graph, and $t_e = \{4, 5, 6, 7, 8\}$ composes the second temporal graph, until $t_e = \{31, 32, 33, 34, 35\}$ composes the eleventh temporal graph. For each graph, we split it into 5 snapshots, i.e., $t_s \in \{1, 2, 3, 4, 5\}$. To be more specific, first, we follow the edge timestamp even distribution, such that one edge timestamp solely occupies one snapshot timestamp; Second, if there are no interactions (i.e., no edges) at a certain edge timestamp, we alternatively split the graph into 5 snapshots based on the total number of edges evenly distributed into 5 snapshots.

The four cross-validation groups are listed as follows.

1. $\mathcal{D}^{train}$ = {Babu, Breitkreutz, Gavin, Hazbun, Ho, Ito, Krogan-LCMS, Krogan-MALDI},
   $\mathcal{D}^{test}$ = {Lambert, Tarassov, Uetz, Yu}
2. $\mathcal{D}^{train}$ = {Breitkreutz, Ho, Krogan-MALDI, Tarassov, Yu, Gavin, Ito, Babu,},
   $\mathcal{D}^{test}$ = {Krogan-LCMS, Hazbun, Lambert, Uetz}
3. $\mathcal{D}^{train}$ = {Babu, Breitkreutz, Ho, Hazbun, Krogan-MALDI, Lambert, Tarassov, Uetz,},
   $\mathcal{D}^{test}$ = {Yu, Gavin, Ito, Krogan-LCMS}
4. $\mathcal{D}^{train}$ = {Babu, Hazbun, Lambert, Tarassov, Yu, Gavin, Uetz, Krogan-LCMS},
   $\mathcal{D}^{test}$ = {Breitkreutz, Ho, Krogan-MALDI, Ito}

D PREPROCESSING OF SOCIAL DATA

In the social network data, we adopt the given timestamp as edge timestamp t_e, such that every temporal graph in each class shares the same range of continuous timestamp t_e. In each social temporal graph, all edge timestamps t_e are evenly divided into 5 snapshots, $t_s \in \{1, 2, 3, 4, 5\}$, based on the ascending order of t_e. Again, if there are no interactions (i.e., no edges) at a certain edge timestamp, we alternatively split the graph into 5 snapshots based on the total number of edges evenly distributed.

We separate the dataset into five cross-validation groups as follows by randomly shuffling the classes of online (for meta-training) and offline (for meta-testing) social networks, respectfully.

1. $\mathcal{D}^{train}$ = {dblp_1, dblp_0, facebook_1, facebook_0, tumblr_1, tumblr_0},
   $\mathcal{D}^{test}$ = {highschool_1, highschool_0, infectious_1, infectious_0, mit_1, mit_0}
2. $\mathcal{D}^{train}$ = {dblp_1, facebook_0, tumblr_1, tumblr_0, dblp_0, facebook_1},
   $\mathcal{D}^{test}$ = {highschool_1, highschool_0, infectious_1, infectious_0, mit_1, mit_0}
3. $\mathcal{D}^{train}$ = {dblp_1, dblp_0, facebook_1, facebook_0, tumblr_1, tumblr_0},
   $\mathcal{D}^{test}$ = {highschool_1, infectious_0, mit_1, highschool_0, infectious_1, mit_0}
4. $\mathcal{D}^{train}$ = {facebook_1, facebook_0, tumblr_1, tumblr_0, dblp_1, dblp_0},
   $\mathcal{D}^{test}$ = {highschool_1, infectious_0, mit_1, highschool_0, infectious_1, mit_0}
5. $\mathcal{D}^{train}$ = {facebook_1, facebook_0, tumblr_1, tumblr_0, dblp_1, dblp_0},
   $\mathcal{D}^{test}$ = {highschool_1, infectious_0, mit_1, highschool_0, mit_0, infectious_1}

E ABLATION STUDY

In Temp-GFSM, we design the multi-time attention mechanism to encode the temporal graph evolution patterns into the representation vector, which has three components: node-level lifelong attention, intra-snapshot attention, and inter-snapshot attention. In the ablation study, we aim to remove them individually to investigate the effectiveness of each part. Thus, we design the following ablation experiment on the biological temporal graph data.

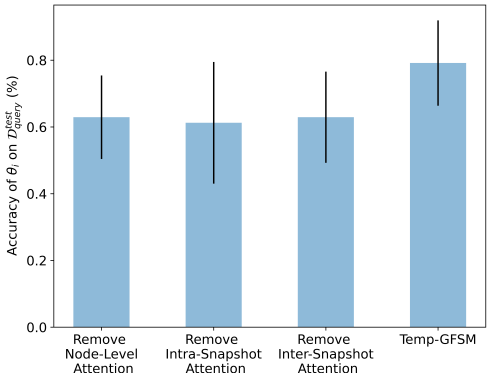


Figure 6: Variants of Temp-GFSM for Temporal Graph Classifications in the 3 Way - 3 Shot Setting.

To ablate the node-level lifelong attention, we remove Eq. 3 directly, such that each node could only look for its previous neighbors but omit the corresponding interval time information. To ablate the intra-snapshot attention, we eliminate the snapshot reconstruction loss function (i.e., Eq. 7) by setting $\gamma = 0$. Then, there will be no constraints on the intra-snapshot level. To remove the inter-snapshot attention, we replace Eq. 9 with the average pooling scheme, such that each snapshot will contribute equally to form the temporal graph representation vector. After we remove each component respectively, in Figure 6, we report the dynamic protein-protein interaction networks classification accuracy in the 3-way 3-shot setting with the same hyperparameters mentioned in Appendix G. As shown in Figure 6, we can observe that each proposed component plays its own role in improving the metric learning ability to help the temporal graph classification problem. Moreover, when we ablate the intra-snapshot attention scheme, the ablated Temp-GFSM achieves the worst performance with the larger variance. It suggests that the snapshot reconstruction loss (i.e., capturing episodic patterns in the streaming graph) is important in identifying the property of temporal graphs for the accurate and stable classification performance. The above observation means that adding the snapshot reconstruction loss $\mathcal{L}_{rec}$ is vital, but to what extent should we pay attention to the snapshot reconstruction loss $\mathcal{L}_{rec}$ in the streaming graph is not clear. Next, we design the parameter analysis in terms of different values of γ.

F PARAMETER SENSITIVITY

Here, we change the weight of the snapshot reconstruction loss $\mathcal{L}_{rec}$ by changing the value of γ, to investigate the effect of capturing episodic patterns in streaming graphs. In Figure 7, we show the dynamic protein-protein interaction networks classification accuracy in the 3-way 1-shot setting with different γ. Other hyperparameters are consistent with Appendix G. As shown in Figure 7, we can see that neither trivial nor dominant snapshot reconstruction loss $\mathcal{L}_{rec}$ is ideal such as $\gamma = 0.3$ or $\gamma = 1.5$, because when the proportion of loss $\mathcal{L}_{rec}$ is too small or too large, the accuracy of Temp-GFSM is not that high and stable. Also, we can see that when the loss $\mathcal{L}_{rec}$ has considerable attention weights like $\gamma = 0.7$, Temp-GFSM could classify accurately and robustly.

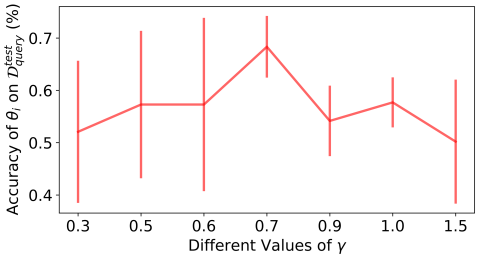


Figure 7: Effectiveness of Temp-GFSM with Different γ in the 3-Way 1-Shot Setting.

G IMPLEMENTATION DETAILS

Our implementation is released here [5]. For the graph kernel methods, the implementation [49] can be found here [6]. For the graph metric learning and graph representation learning algorithms, we set the dimension of graph representation vectors as 64, and the implementation [6, 44, 59, 61] can be found here [7] [8] [9] [10]. As for the performance of our Temp-GFSM method in Table 3 and Tabel 4, we set the learning rate α and β as 0.001, and the weight for the snapshot reconstruction loss $\gamma = 0.7$. The experiments are performed on a Linux machine with a single NVIDIA Tesla V100 32GB GPU.

[5] https://github.com/DongqiFu/Temp-GFSM
[6] https://ysig.github.io/GraKeL/0.1a8/documentation/introduction.html
[7] https://github.com/moranbel/tdGraphEmbed
[8] https://github.com/benedekrozemberczki/karateclub
[9] https://github.com/snap-stanford/CAW
[10] https://github.com/StatsDLMathsRecomSys/Inductive-representation-learning-on-temporal-graphs