A Data-Driven Graph Generative Model for Temporal Interaction Networks

Dawei Zhou, Lecheng Zheng, Jiawei Han, and Jingrui He
University of Illinois at Urbana-Champaign, Urbana, IL, USA
{dzhou21, lecheng4, hanj, jingrui}@illinois.edu

ABSTRACT

Deep graph generative models have recently received a surge of attention due to its superiority of modeling realistic graphs in a variety of domains, including biology, chemistry, and social science. Despite the initial success, most, if not all, of the existing works are designed for static networks. Nonetheless, many realistic networks are intrinsically dynamic and presented as a collection of system logs (i.e., timestamped interactions/edges between entities), which pose a new research direction for us: how can we synthesize realistic dynamic networks by directly learning from the system logs? In addition, how can we ensure the generated graphs preserve both the structural and temporal characteristics of the real data?

To address these challenges, we propose an end-to-end deep generative framework named *TagGen*. In particular, we start with a novel sampling strategy for jointly extracting structural and temporal context information from temporal networks. On top of that, *TagGen* parameterizes a bi-level self-attention mechanism together with a family of local operations to generate temporal random walks. At last, a discriminator gradually selects generated temporal random walks, that are plausible in the input data, and feeds them to an assembling module for generating temporal networks. The experimental results in seven real-world data sets across a variety of metrics demonstrate that (1) *TagGen* outperforms all baselines in the temporal interaction network generation problem, and (2) *TagGen* significantly boosts the performance of the prediction models in the tasks of anomaly detection and link prediction.

CCS CONCEPTS

• **Networks** → **Topology analysis and generation**; • **Theory of computation** → **Dynamic graph algorithms**.

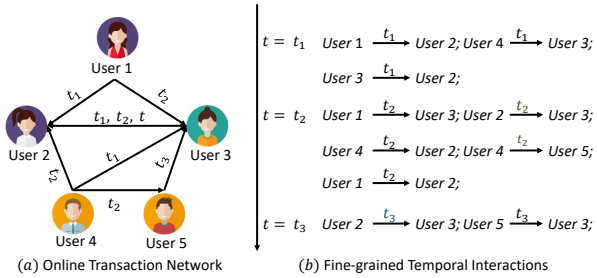
KEYWORDS

Graph Generative Model, Temporal Networks, Transformer Model

ACM Reference Format:
Dawei Zhou, Lecheng Zheng, Jiawei Han, and Jingrui He. 2020. A Data-Driven Graph Generative Model for Temporal Interaction Networks. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3394486.3403082

KDD '20, August 23–27, 2020, Virtual Event, CA, USA

ACM ISBN 978-1-4503-7998-4/20/08…$15.00
https://doi.org/10.1145/3394486.3403082



(a) Online Transaction Network (b) Fine-grained Temporal Interactions

Figure 1: An example of *temporal interaction networks*. (a) An online transaction network with five users. (b) The corresponding system logs presented in the form of time-stamped edges between users.

1 INTRODUCTION

Graph presents a fundamental abstraction for modeling complex systems in a variety of domains, ranging from chemistry [39], security [4, 16, 42], recommendation [25, 33], and social science [34]. Therefore, mimicking and generating realistic graphs have been extensively studied in the past. The traditional graph generative models are mostly designed to model a particular family of graphs based on some specific structural assumptions, such as heavy-tailed degree distribution [3], small diameter [10], local clustering [38], etc. In addition to the traditional graph generative models, a surge of research efforts on deep generative models [12, 17] have been recently observed in the task of graph generation. These approaches [5, 40] are trained directly from the input graphs without incorporating prior structural assumptions and often achieve promising performance in preserving diverse network properties of real networks.

Despite the initial success of deep generative models on graphs, most of the existing techniques are designed for static networks. Nonetheless, many real networks are intrinsically dynamic and stored as a collection of system logs (i.e., timestamped edges between entities). For example, in Fig. 1, an online transaction network can be intrinsically presented as a sequence of timestamped edges (i.e., financial transactions) between users. When an online transaction is completed, a system log file (i.e., a timestamped edge from one account to another) will be automatically generated and stored in the system. A conventional way of modeling such dynamic systems is to construct time-evolving graphs [36, 44] by aggregating timestamps into a sequence of snapshots. One drawback comes from the uncertainty of defining the proper resolution of the time-evolving graphs. If the resolution is too fine, the massive number of snapshots will bring intractable computational cost when training deep generative models; if the resolution is too coarse, fine-grained temporal context information (e.g., the addition/deletion of nodes and edges) might be lost during the time aggregation.

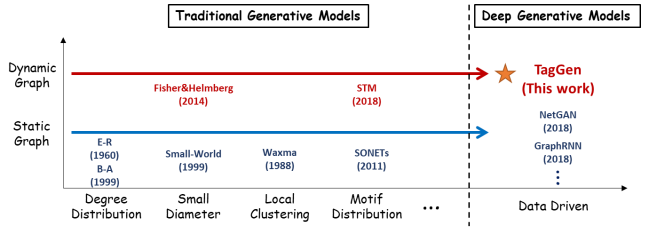


Figure 2: A two-dimensional conceptual space of graph generative models.

Fig. 2 compares various graph generators in a two-dimensional conceptual space in order to demonstrate the limitation of existing techniques as compared to ours. In this paper, for the first time, we aim to address the following three open challenges: (*Q.1*) Can we directly learn from the raw temporal networks (i.e., temporal interaction network) represented as a collection of timestamped edges (see Fig. 1 (b)) instead of constructing the time-evolving graphs? (*Q.2*) Can we develop an end-to-end deep generative model that can ensure the generated graphs preserve the structural and temporal characteristics (e.g., the heavy tail of degree distribution, and shrinking network diameter over time) of the original data?

To this end, we propose *TagGen*, a deep graph generative model for *temporal interaction networks* to tackle all of the aforementioned challenges. We first propose a random walk sampling strategy to jointly extract the key structural and temporal context information from the input graphs. On top of that, we develop a bi-level self-attention mechanism which can be directly trained from the extracted temporal random walks while preserving temporal interaction network properties. Moreover, we designed a novel network context generation scheme, which defines a family of local operations to perform *addition* and *deletion* of nodes and edges, thus mimicking the evolution of real dynamic systems. In particular, *TagGen* maintains the state of the graph and generates new temporal edges by training from the extracted temporal random walks [27]; the *addition* operation randomly chooses a node to be connected with another one at a timestamp t; the *deletion* operation randomly terminates the interaction between two nodes at timestamp t; all the proposed operations are either accepted or rejected by a discriminator module in *TagGen* based on the current states of the constructed graph. At last, the selected plausible temporal random walks will be fed into an assembling module to generate temporal networks.

The main contributions of this paper are summarized below.

- **Problem.** We formally define the problem of *temporal interaction network* generation and identify its unique challenges arising from real applications.
- **Algorithm.** We propose an end-to-end learning framework for *temporal interaction network* generation, which can (1) directly learn from a series of timestamped nodes and edges and (2) preserve the structural and temporal characteristics of the input data.
- **Evaluations.** We perform extensive experiments and case studies on seven real data sets, showing that *TagGen* achieves superior performances compared with the previous methods in the tasks of temporal graph generation and data augmentation.

The rest of our paper is organized as follows. Problem definition is introduced in Section 2, followed by the details of our proposed framework *TagGen* in Section 3. Experimental results are reported in Section 4. In Section 5, we review the existing literature before we conclude the paper in Section 6.

2 PROBLEM DEFINITION

The main symbols used in this paper are summarized in Table 1 of Appendix A. We formalize the graph generation problem for temporal interaction networks [21, 27, 29], and present our learning problem with inputs and outputs. Different from conventional dynamic graphs that are defined as a sequence of discrete snapshots, the temporal interaction network is represented as a collection of temporal edges. Each node is associated with multiple timestamped edges at different timestamps, which results in the different occurrences of node $v = \{v^{t_1}, \ldots, v^T\}$. For example, in Fig. 3, the node v_a is associated with three occurrences $\{v_a^{t_1}, v_a^{t_2}, v_a^{t_3}\}$ that appear at timestamps t_1, t_2 and t_3. The formal definitions of temporal occurrence and temporal interaction network are given as follows.

Definition 1 (**Temporal Occurrence**). *In a temporal interaction network, a node v is associated with a bag of temporal occurrences $v = \{v^{t_1}, v^{t_2}, \ldots\}$, which instance the occurrences of node v at timestamps $\{t_1, t_2, \ldots\}$ in the network.*

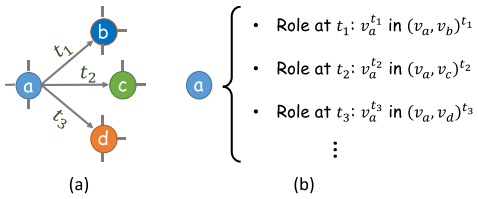


Figure 3: An example of node v_a and its temporal occurrences. (a) A miniature of a temporal interaction network. (b) The occurrences of node v_a that appear at t_1, t_2 and t_3.

Definition 2 (**Temporal Interaction Network**). *A temporal interaction network $\widetilde{G} = (\widetilde{V}, \widetilde{E})$ is formed by a collection of nodes $\widetilde{V} = \{v_1, v_2, \ldots, v_n\}$ and a series of timestamped edges $\widetilde{E} = \{e_1^{t_{e_1}}, e_2^{t_{e_2}}, ..., e_m^{t_{em}}\}$, where $e_i^{t_{e_i}} = (u_{e_i}, v_{e_i})^{t_{e_i}}$.*

In the static setting, existing works [30] define the network neighborhood $\mathcal{N}(v)$ of node v as a set of nodes that are generated through some neighborhood sampling strategies. Here, we generalize the notion of network neighborhood to the temporal interaction network setting as follows.

Definition 3 (**Temporal Network Neighborhood**). *Given a temporal occurrence v^{t_v} at timestamp t_v, the neighborhood of v^{t_v} is defined as $\mathcal{N}_{FT}(v^{t_v}) = \{v_i^{t_{v_i}} | f_{sp}(v_i^{t_{v_i}}, v^{t_v}) \leq d_{\mathcal{N}_{FT}}, |t_v - t_{v_i}| \leq t_{\mathcal{N}_{FT}}\}$, where $f_{sp}(\cdot|\cdot)$ denotes the shortest path between two nodes, $d_{\mathcal{N}_{FT}}$ is the user-defined neighborhood range, and $t_{\mathcal{N}_{FT}}$ refers to the user-defined neighborhood time window.*

In [27], the authors define the notion of *Temporal Walk*, which is presented as a sequence of vertices following a time-order constraint. In this paper, we relax such a constraint by considering

that all the nodes within a neighborhood time window $[t_v - t_{\mathcal{N}_{FT}} + 1, t_v + t_{\mathcal{N}_{FT}}]$ are the temporal neighbors of v^{t_v} and can be accessed from v via a random walk. Here, we formally define the k-Length Temporal Walk as follows.

Definition 4 (k-**Length Temporal Walk**). *Given a temporal interaction network $\widetilde{G}$, a k-length temporal walk $W = \{w_1, \ldots, w_k\}$ is defined as a sequence of incident temporal walks traversed one after another, i.e., $w_i = (u_{w_i}, v_{w_i})^{t_{w_i}}$, $i = 1, \ldots, k$, where u_{w_i} and v_{w_i} are the source node and destination node of the i^{th} temporal walk w_i in W respectively.*

With all the aforementioned notions, we are ready to formalize the temporal interaction network generation problem as follows.

Problem 1. ***Temporal Interaction Network Generation***

Input: *a temporal interaction network $\widetilde{G}$, which is presented as a collection of timestamped edges $\{(u_{e_1}, v_{e_1})^{t_{e_1}}, \ldots, (u_{e_m}, v_{e_m})^{t_{e_m}}\}$.*

Output: *a synthetic temporal interaction network $\widetilde{G}' = (\widetilde{V}', \widetilde{E}')$ that accurately captures the structural and temporal properties of the observed temporal network $\widetilde{G}$.*

3 MODEL

In this section, we introduce *TagGen*, a graph generative model for temporal interaction networks. The core idea of *TagGen* is to train a bi-level self-attention mechanism together with a family of local operations to model and generate temporal random walks for assembling temporal interaction networks. In particular, we first introduce the overall learning framework of *TagGen*. Then, we discuss the technical details of *TagGen* regarding context sampling, sequence generation, sample discrimination, and graph assembling in temporal interaction networks. At last, we present an end-to-end optimization algorithm for training *TagGen*.

3.1 A Generic Learning Framework

An overview of our proposed framework is presented in Fig. 4, which consists of four major stages. Given a temporal interaction network defined by a collection of temporal edges (i.e., timestamped interactions), we first extract network context information of temporal interaction networks by sampling a set of temporal random walks [27] via a novel sampling strategy. Second, we develop a deep generative mechanism, which defines a set of simple yet effective operations (i.e., addition and deletion over temporal edges) to generate synthetic random walks. Third, a discriminator is trained over the sampled temporal random walks to determine whether the generated temporal walks follow the same distributions as the real ones. At last, we generate temporal interaction network, by collecting the qualified synthetic temporal walks via the discriminator. In the following subsections, we describe each stage of *TagGen* in details.

Context sampling. Inspired by the advances of network embedding approaches [30], we view the problem of temporal network context sampling as a form of local exploration in network neighborhood $\mathcal{N}_{FT}$ via temporal random walks [27]. Specifically, given a temporal occurrence v^{t_v}, we aim to extract a set of sequences that are capable of generating its neighborhood $\mathcal{N}_{FT}(v^{t_v})$. Notice that in order to fairly and effectively sample neighborhood context, we should select the most representative temporal occurrences to serve as initial nodes from the entire data. Here we propose to estimate the *context importance* via computing the conditional probability $p(v^{t_v}|\mathcal{N}_{FT}(v^{t_v}))$ of each temporal occurrence v^{t_v} given its temporal network neighborhood context $\mathcal{N}_{FT}(v^{t_v})$ as follows.

$$p(v^{t_v}|\mathcal{N}_{FT}(v^{t_v})) = p(v^{t_v}|\mathcal{N}_S(v^{t_v}), \mathcal{N}_T(v^{t_v})) \tag{1}$$

where $\mathcal{N}_T(v^{t_v})$ and $\mathcal{N}_S(v^{t_v})$ denote the temporal neighborhood and structural neighborhood of v^{t_v} respectively.

$$\mathcal{N}_T(v^{t_v}) = \{v_i^{t_{v_i}} || t_v - t_{v_i}| \leq t_{\mathcal{N}_{FT}}\}$$

$$\mathcal{N}_S(v^{t_v}) = \{v_i^{t_{v_i}} | f_{sp}(v_i^{t_{v_i}}, v^{t_v}) \leq d_{\mathcal{N}_{FT}}\}$$

Intuitively, when $p(v^{t_v}|\mathcal{N}_{FT}(v^{t_v}))$ is high, it turns out that v^{t_v} is a representative node in its neighborhood, which could be a good initial point for random walks; when $p(v^{t_v}|\mathcal{N}_{FT}(v^{t_v}))$ is low, it is highly possible that $p(v^{t_v})$ is an outlier point, whose behaviors deviate from its neighbors. A key challenge here is how to estimate $p(v^{t_v}|\mathcal{N}_{FT}(v^{t_v}))$. If $p(v^{t_v}|\mathcal{N}_T(v^{t_v}))$ and $p(v^{t_v}|\mathcal{N}_S(v^{t_v}))$ are independent to each other, it is easy to see

$$p(v^{t_v}|\mathcal{N}_{FT}(v^{t_v})) = p(v^{t_v}|\mathcal{N}_T(v^{t_v}))p(v^{t_v}|\mathcal{N}_S(v^{t_v})) \tag{2}$$

where $p(v^{t_v}|\mathcal{N}_T(v^{t_v}))$ and $p(v^{t_v}|\mathcal{N}_S(v^{t_v}))$ can be estimated via some heuristic methods [27, 30]. However, in real networks, the topology context and temporal context are correlated to some extend, which has been observed in [7]. For instance, the high-degree nodes (i.e., $p(v^{t_v}|\mathcal{N}_S(v^{t_v}))$ is high) have a high probability to be active in a future timestamp (i.e., $p(v^{t_v}|\mathcal{N}_T(v^{t_v}))$ is high) , and vice versa. These observations allow us to state a weak dependence [1] between the topology neighborhood distribution and temporal neighborhood distribution.

Definition 5 (**Weak Dependence**). *For any $v^{t_v} \in \widetilde{V}$, the corresponding temporal neighborhood distribution $p(v^{t_v}|\mathcal{N}_T(v^{t_v}))$ and topology neighborhood distribution $p(v^{t_v}|\mathcal{N}_S(v^{t_v}))$ are weakly dependent on each other, such that, for $\delta > 0$,*

$$p(v^{t_v}|\mathcal{N}_{FT}(v^{t_v})) \geq \delta[p(v^{t_v}|\mathcal{N}_T(v^{t_v}))p(v^{t_v}|\mathcal{N}_S(v^{t_v}))].$$

Based on Def. 5, here we establish the relationship between $p(v^{t_v}|\mathcal{N}_{FT}(v^{t_v}))$ and $p(v^{t_v}|\mathcal{N}_T(v^{t_v}))$, $p(v^{t_v}|\mathcal{N}_S(v^{t_v}))$.

Lemma 1. *For any $v^{t_v} \in \widetilde{V}$, if the temporal neighborhood distribution $p(v^{t_v}|\mathcal{N}_T(v^{t_v}))$ and topology neighborhood distribution $p(v^{t_v}|\mathcal{N}_S(v^{t_v}))$ are weakly dependent on each other, then the following inequality holds:*

$$p(v^{t_v}|\mathcal{N}_{FT}(v^{t_v})) \geq \alpha \frac{p(v^{t_v}|\mathcal{N}_S(v^{t_v}))p(v^{t_v}|\mathcal{N}_T(v^{t_v}))p(\mathcal{N}_S(v^{t_v}))p(\mathcal{N}_T(v^{t_v}))}{p(\mathcal{N}_S(v^{t_v}), \mathcal{N}_T(v^{t_v}))} \tag{3}$$

where $\alpha = \frac{\delta}{p(v^{t_v})}$.

The proof of this Lemma can be found in Appendix B. Following [30], we assume $p(v^{t_v}|\mathcal{N}_S(v^{t_v}))$ and $p(v^{t_v}|\mathcal{N}_T(v^{t_v}))$ follow a uniform distribution, where all the temporal entities in a local region are equally important. Then, by computing $p(\mathcal{N}_S(v^{t_v}))$, $p(\mathcal{N}_T(v^{t_v}))$ and $p(\mathcal{N}_S(v^{t_v}), \mathcal{N}_T(v^{t_v}))$ (e.g., via kernel density estimation approaches [35]), we can infer the *context importance* $p(v^{t_v}|\mathcal{N}_{FT}(v^{t_v}))$ based on Eq. 3 for selecting initial nodes.

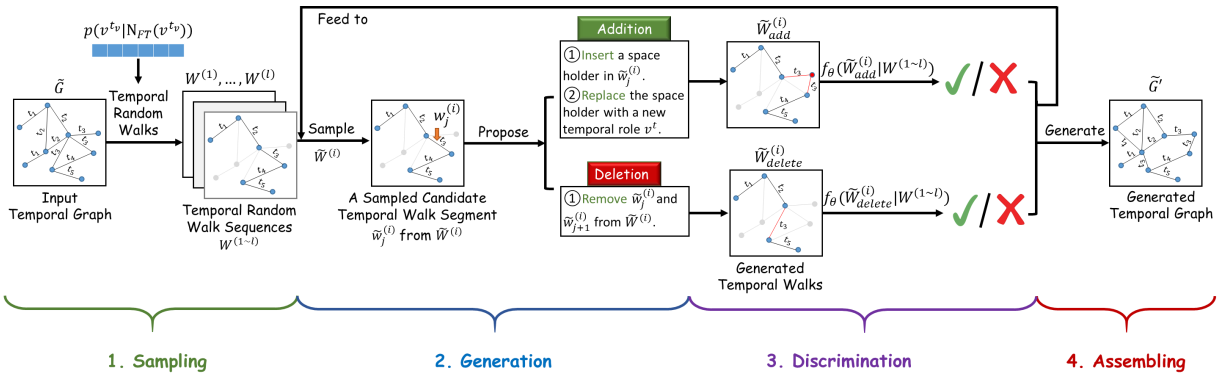


Figure 4: The proposed *TagGen* framework.

After selecting the initial temporal occurrence, we use the biased temporal random walk [27] to extract a collection of temporal walks for training *TagGen*. The key reasons for using random walk based sampling approaches are their flexibility of controlling sequence length and the capability of jointly capturing structural and temporal neighborhood context information, as shown in [15, 27, 30].

Sequence generation. To generate the synthetic temporal random walks, a straightforward solution is to train a sequence model by learning from the extracted random walks [5]. However, in the temporal network setting, it is unclear how to mimic the network evolution and produce temporal interaction networks. Therefore, in this paper, we design a family of local operations, i.e., Action = $\{add, delete\}$, to perform *addition* and *deletion* of temporal entities and mimic the evolution of real dynamic networks. In particular, given a k-length temporal random walk $\widetilde{W}^{(i)} = \{\widetilde{w}_1^{(i)}, \ldots, \widetilde{w}_k^{(i)}\}$, we first sample a candidate temporal walk segment $\widetilde{w}_j^{(i)} \in \widetilde{W}^{(i)}$ following a user-defined prior distribution $p(\widetilde{W}^{(i)})$. In this paper, we assume $p(\widetilde{W}^{(i)})$ follows a uniform distribution, although the proposed techniques can be naturally extended to other types of prior distribution. Then, we randomly perform one of the following operations with probability $p_{action} = \{p_{add}, p_{delete}\}$.

- *add* : The *add* operation is done in a two-step fashion. First, we insert a place holder token in the candidate temporal walk segment $\widetilde{w}_j^{(i)} = (u_{\widetilde{w}_j^{(i)}}, v_{\widetilde{w}_j^{(i)}})^{t_{\widetilde{w}_j^{(i)}}}$, and then replace a new temporal entity $v*^{t_{v*}}$ with the place holder token such that $\widetilde{w}_j^{(i)}$ is broken into $\{(u_{\widetilde{w}_j^{(i)}}, v*)^{t_{v*}}, (v*, v_{\widetilde{w}_j^{(i)}})^{t_{\widetilde{w}_j^{(i)}}}\}$. The length of the modified temporal random walk sequence $\widetilde{W}_{add}^{(i)}$ would be $k+1$.
- *delete* : The *delete* operation removes the candidate temporal walk segment $\widetilde{w}_j^{(i)}$ from $\widetilde{W}^{(i)} = \{\widetilde{w}_1^{(i)}, \ldots, \widetilde{w}_j^{(i)}, \ldots, \widetilde{w}_k^{(i)}\}$, such that the length of the modified temporal random walk $\widetilde{W}_{delete}^{(i)}$ would be $k-1$.

Sample discrimination. To ensure the generated graph context follows the similar global structure distribution as the input, *TagGen* is equipped with a discriminator model $f_\theta(\cdot)$, which

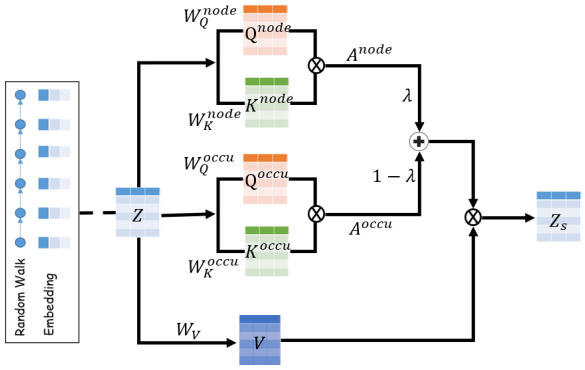


Figure 5: Bi-level self-attention.

aims to distinguish whether the generated temporal networks follow the same distribution as the original graphs. For each generated temporal random walk $\widetilde{W}_{action}^{(i)}$ after a certain operation $action = \{add, delete\}$, *TagGen* computes the conditional probability $p(\widetilde{W}_{action}^{(i)} | W^{(1\sim l)})$ given the extracted real temporal random walks $W^{(1\sim l)} = \{W^{(1)}, \ldots, W^{(l)}\}$ as follows.

$$p(\widetilde{W}_{action}^{(i)} | W^{(1\sim l)}) \propto p_{action}(action) f_\theta(\widetilde{W}_{action}^{(i)}) \tag{4}$$

where $f_\theta(\cdot)$ computes the likelihood of observing $\widetilde{W}_{action}^{(i)}$ given the training data $W^{(1\sim l)} = \{W^{(1)}, \ldots, W^{(l)}\}$; $p_{action}(action)$ weights the proposed operation over $\widetilde{W}_{action}^{(i)}$.

Some recent graph generative frameworks (e.g., [5, 40]) model the extracted graph sequences via recurrent neural networks (RNNs) or long short-term memory (LSTM) architectures. However, such sequential nature inherently prevents parallelism and results in intractable running time for long sequence length [37]. For instance, GraphRNN [40] requires to map the n-node graph into length-n sequences for training purposes. Inspired by the recent advances of Transformer models in nature language processing [37], we propose to employ self-attention mechanisms to impose global dependencies among temporal entities (i.e., nodes and temporal occurrences) and reduce the overall sequential computation load. However, direct implementation with standard Transformer parameterization may fail to capture such bi-level dependencies (i.e.,

node-level dependencies and occurrence-level dependencies). Here, we propose a bi-level self-attention mechanism illustrated in Fig. 5. In particular, given a k-length temporal random walk $\tilde{W}^{(i)}$, we first obtain the d-dimensional representation $Z \in \mathbb{R}^{n \times d}$ for each v^t (i.e., node v at timestamp t) via temporal network embedding approaches, e.g., [27]. As each node v is naturally represented as a bag of temporal occurrences $v = \{v^{t1}, v^{t2}, \ldots, v^T\}$, the bi-level self-attention mechanism is designed to jointly learn (1) the dependencies among nodes in $\tilde{G}$ and (2) the dependencies among different temporal occurrences. Following the notations in [37], we define the occurrence-level attention $\boldsymbol{A^{occu}} \in \mathbb{R}^{n_r \times n_r}$ and node-level attention $\boldsymbol{A^{node}} \in \mathbb{R}^{n_r \times n_r}$ as follows.

$$\boldsymbol{A^{occu}}(v_i^{t_1}, v_j^{t_2}) = \frac{(\boldsymbol{z_i^{t_1}} \boldsymbol{W_Q^{occu}}) \odot (\boldsymbol{z_j^{t_2}} \boldsymbol{W_K^{occu}})}{\sqrt{d_k}} \tag{5}$$

$$\boldsymbol{A^{node}}(v_i^{t_1}, v_j^{t_2}) = \frac{(f_{agg}(\boldsymbol{z_i^{t_1}}) \boldsymbol{W_Q^{node}}) \odot (f_{agg}(\boldsymbol{z_j^{t_2}}) \boldsymbol{W_K^{node}})}{\sqrt{d_k}} \tag{6}$$

where $z_i^{t_1}(z_i^{t_2}) \in \mathbb{R}^{1 \times d}$ is the d-dimensional embedding of node $v_i^{t_1}(v_i^{t_2})$; $\boldsymbol{W_Q^{occu}} \in \mathbb{R}^{d \times d_k}$ and $\boldsymbol{W_K^{occu}} \in \mathbb{R}^{d \times d_k}$ are the occurrence-level query weight matrix and key weight matrix, respectively; similarly, $\boldsymbol{W_Q^{node}} \in \mathbb{R}^{d \times d_k}$ and $\boldsymbol{W_K^{node}} \in \mathbb{R}^{d \times d_k}$ are the node-level query weight matrix and key weight matrix, respectively; d_k is a scaling factor; $f_{agg}(\cdot)$ is an aggregation function that summarizes all the occurrence-level information for each node. For implementation purposes, we define $f_{agg}(v_i^t) = \sum_{v_i^t \in v_i} \boldsymbol{z_i^t}$, such that $f_{agg}(v_i^{t_1}) = f_{agg}(v_i^{t_2})$ when $t_1 \neq t_2$. In this way, the entries (i.e., rows) in $\boldsymbol{A^{occu}}$ and $\boldsymbol{A^{node}}$ are exactly aligned. Moreover, we introduce a coefficient $\lambda \in [0, 1]$ to balance the occurrence-level attention and node-level attention and obtain the final bi-level self-attention $\boldsymbol{Z_s}$ as follows.

$$\boldsymbol{Z_s} = [\lambda \times \text{softmax}(\boldsymbol{A^{node}}) + (1 - \lambda) \times \text{softmax}(\boldsymbol{A^{occu}})]\boldsymbol{V} \tag{7}$$

where $\boldsymbol{V} = \boldsymbol{W_V Z}$ and $\boldsymbol{W_V}$ denotes the value weight matrix.

With the single head attention described in Fig. 5, we employ $h = 4$ parallel attention layers (i.e., heads) in discriminator $f_\theta(\cdot)$ for selecting the qualified synthetic random walks $\widetilde{W}_{action}^{(i)}$. The update rule of the hidden representations in $f_\theta(\cdot)$ is the same as the standard Transformer model defined in [37]. At the end of the stage 3, all of the selected synthetic temporal random walks via the $f_\theta(\cdot)$ will be fed to the beginning of Stage 2 (see Fig. 4) to gradually modify these sequences until the user-defined stopping criteria are met and the sequences are ready for assembling (Stage 4).

Graph assembling. In the previous stage, we generate synthetic temporal random walks by gradually performing local operations on the extracted real temporal random walks. In this stage, we assemble all the generated temporal random walks and generate the temporal interaction networks. In particular, we first compute the frequency counts $s(e^{t_e})$ of each temporal edge $e^{t_e} = (u, v)^{t_e}$ in the generated temporal random walks. To ensure the frequency counts are reliable, we use a larger number of the extract temporal random walks from the original graphs to avoid the case where some unrepresented temporal occurrences (i.e., with a small degree) are not sampled. In order to transform these frequency counts to discrete temporal edges, we use the following strategies: (1) we firstly generate at least one temporal edge starting from each temporal occurrence v^{t_v} with probability $p(v^{t_v}, v^* \in \mathcal{N}_S(v^{t_v})) = \frac{s(e^{te}=(v,v^*)^{t_v})}{\sum_{v^* \in \mathcal{N}_S(v^{t_v})} s(e^{te}=(v,v^*)^{t_v})}$ to ensure all the observed temporal occurrences in $\widetilde{G}$ are included; (2) then we generate at least one temporal edge at each timestamp with probability $p(e^{t_e}) = \frac{s(e^{t_e})}{\sum_{e_i^{t_{e_i}}} s(e_i^{t_{e_i}})}$; (3) we generate the temporal edges with the largest counts until the generated graph has the same edge density as the original one.

Algorithm 1 The *TagGen* Learning Framework.

Input:

Temporal interaction network $\widetilde{G}$ and parameters including neighborhood range $d_{\mathcal{N}_{FT}}$, neighborhood time window $t_{\mathcal{N}_{FT}}$, number of initial node l, walks per initial temporal occurrences γ, walk length k and constants c_1 and $\xi \in (0.5, 1)$.

Output:

Synthetic temporal interaction network $\widetilde{G'}$;

1: Sample l initial temporal occurrences based on Eq. 3.
2: Sample γ temporal random walks starting from each initial temporal occurrence with neighborhood range $d_{\mathcal{N}_{FT}}$ and neighborhood time window $t_{\mathcal{N}_{FT}}$, and store them in $\mathcal{S}$.
3: Train discriminator f_θ based on $\mathcal{S}$.
4: Let $\mathcal{S}' = \{\}$.
5: **for** $i = 1 : \gamma \times l$ **do**
6: Initialize $\widetilde{W}^{(i)}$ with the first entry in $W^{(i)}$, i.e., $\widetilde{W}^{(i)} = \{w_1^{(i)}\}$.
7: **for** $c = 1 : c_1$ **do**
8: Sample a candidate temporal walk segment $\widetilde{w}_j^{(i)}$ from $\widetilde{W}^{(i)}$.
9: Draw a number $random \sim Unif(0, 1)$.
10: If $random < \xi$, perform add operation on $w_j^{(i)}$; if $random \leq \xi$, perform $delete$ operation on $w_j^{(i)}$.
11: If discriminator f_θ approves the proposal $\widetilde{W}_{action}^{(i)}$, replace $\widetilde{W}^{(i)}$ with $\widetilde{W}_{action}^{(i)}$; if not, continue.
12: **end for**
13: Add $\widetilde{W}^{(i)}$ into $\mathcal{S}'$.
14: **end for**
15: Construct $\widetilde{G'}$ based on $\mathcal{S}'$ by ensuring all the temporal occurrences and timestamps are included in $\widetilde{G'}$.

3.2 Optimization Algorithm

To optimize *TagGen*, we use stochastic gradient descent [6] (SGD) to learn the hidden parameters of *TagGen*. The optimization algorithm is described in Alg. 1. The given inputs include the Temporal interaction network $\widetilde{G}$, neighborhood range $d_{\mathcal{N}_{FT}}$, neighborhood time window $t_{\mathcal{N}_{FT}}$, number of initial nodes l, walks per initial nodes γ, walk length k, the number of operations per sequence c_1, and constant parameters $\xi \in (0.5, 1)$. With $\xi > 0.5$, we enforce the number of *add* operation to be larger than the number of *delete* operation. In this way, we can avoid the case of generating zero-entry temporal random walk sequences. From Step 1 to Step 3, we sample a set of temporal random walks $\mathcal{S}$ from the input data and train the discriminator $f_\theta(\cdot)$. Step 4 to Step 14 is the main body of *TagGen*, which generates the exactly sample number of temporal

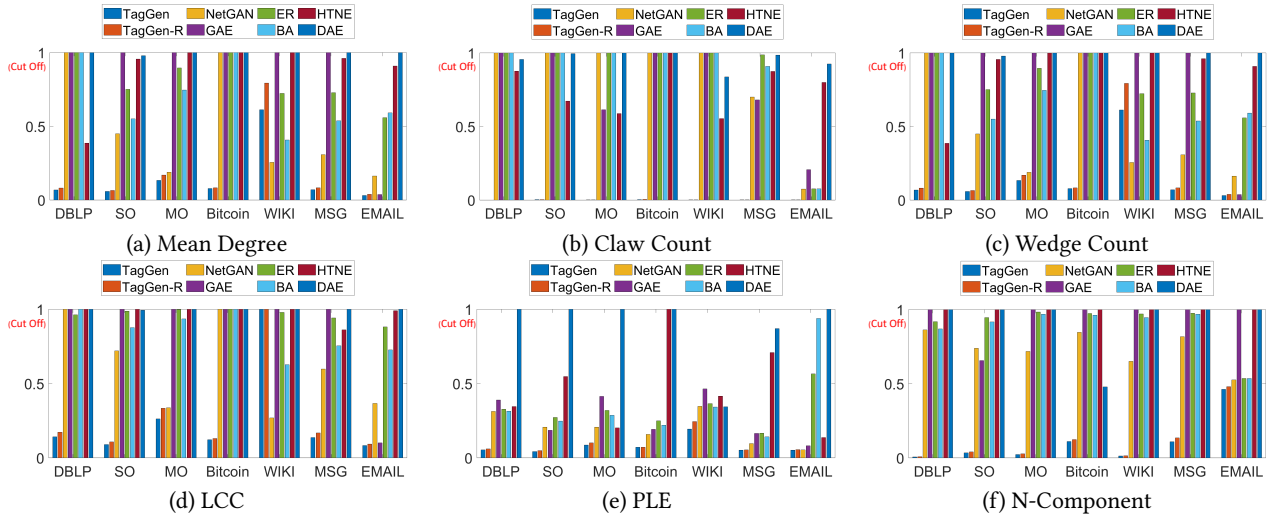


Figure 6: Average score $f_{avg}(\cdot)$ comparison with six metrics across seven temporal networks. Best viewed in color. We cut off high values for better visibility. (Smaller metric values indicate better performance)

random walks as in $\mathcal{S}$. We firstly initial each synthetic walk $\widetilde{W}^{(i)}$ with first entry in $W^{(i)}$, i.e., $\widetilde{W}^{(i)} = \{w_1^{(i)}\}$. From Step 7 to Step 12, we perform c_1 times operations (i.e., *add* and *delete*) to generate context for each synthetic walk $\widetilde{W}^{(i)}$ and use discriminator $f_\theta(\cdot)$ to select the qualified temporal random walks to be stored in $\mathcal{S}'$. In the end, Step 15 constructs the $\widetilde{G'}$ based on $\mathcal{S}'$ by ensuring all the temporal occurrences and timestamps are included in $\widetilde{G'}$ as discussed in the previous subsection regarding Stage 4.

4 EXPERIMENTAL RESULTS

In this section, we demonstrate the performance of our proposed *TagGen* framework across seven real temporal networks in graph generation and data augmentation. Additional results regarding scalability analysis are reported in Appendix E.

4.1 Experiment Setup

Data Sets: We evaluate *TagGen* on seven real temporal networks, including DBLP [43], SO [43], MO [29], WIKI [23], EMAIL [29], MSG [28] and BITCOIN [20]. The statistics of data sets are summarized in Appendix C.

Comparison Methods: We compare *TagGen* with two traditional graph generative models (i.e., Erdös-Rényi (ER) [9] and Barabási-Albert (BA) [3]), two deep graph generative models (GAE [18], NetGAN [5]), and two dynamic graph generators based on temporal network embedding approaches (HTNE [45], DAE [13]). Note that HTNE and GAE are not designed for graph generation. To generate temporal networks, we utilize the learned temporal network embedding to construct the adjacency matrix at each timestamp.

Evaluation Metrics: We consider six widely-used network properties (i.e., Mean Degree, Claw Count, Wedge Count, PLE, LCC, N-Component) for evaluation, which are elaborated in Appendix D. As all of these metrics are designed for static graphs, here we generalize the aforementioned metrics to the dynamic setting in the form of mean value and median value. In particular, given the real network $\widetilde{G}$, the synthetic one $\widetilde{G'}$ and a user-specific metric $f_m(\cdot)$, we first construct a sequence of snapshots $\widetilde{S}^t$ ($\widetilde{S'}^t$), $t = 1, \ldots, T$, of $\widetilde{G}$ ($\widetilde{G'}$) by aggregating from the initial timestamp to the current timestamp t. Then, we measure the averaged/median discrepancy (in percentage) between the original graph and the generated graph in terms of the given metric $f_m(\cdot)$ as follows.

$$f_{avg}(\widetilde{G}, \widetilde{G'}, f_m) = Mean_{t=1:T}(|\frac{f_m(\widetilde{S}^t) - f_m(\widetilde{S'}^t)}{f_m(\widetilde{S}^t)}|)$$

$$f_{med}(\widetilde{G}, \widetilde{G'}, f_m) = Median_{t=1:T}(|\frac{f_m(\widetilde{S}^t) - f_m(\widetilde{S'}^t)}{f_m(\widetilde{S}^t)}|)$$

4.2 Quantitative Results for Graph Generation

We compare *TagGen* with six baseline methods across seven dynamic networks regarding six network property metrics in the form of $f_{avg}(\cdot)$ and $f_{med}(\cdot)$ are shown in Fig. 6 and Fig.7. For the static methods, we apply them on the constructed graph snapshots at each timestamp and then report the results. In all of these figures, the performance is the smaller metric values, the better. For the sake of better visualization, the values of the scores are set to be one if any value is greater than 1. We draw several interesting observations from these results. (1) *TagGen* outperforms the baseline methods across the six evaluation metrics and seven data sets in most of the cases. (2) The random graph algorithms (i.e., ER and BA) perform well (i.e., close to *TagGen* and better than NetGAN and GAE) with Mean Degree (shown in Fig. 6 (a) and Fig. 7 (a)), but perform worse than the competitors with most of the other metrics. This is because such random graph algorithms are often designed to model a certain structural distribution (e.g., degree distribution)

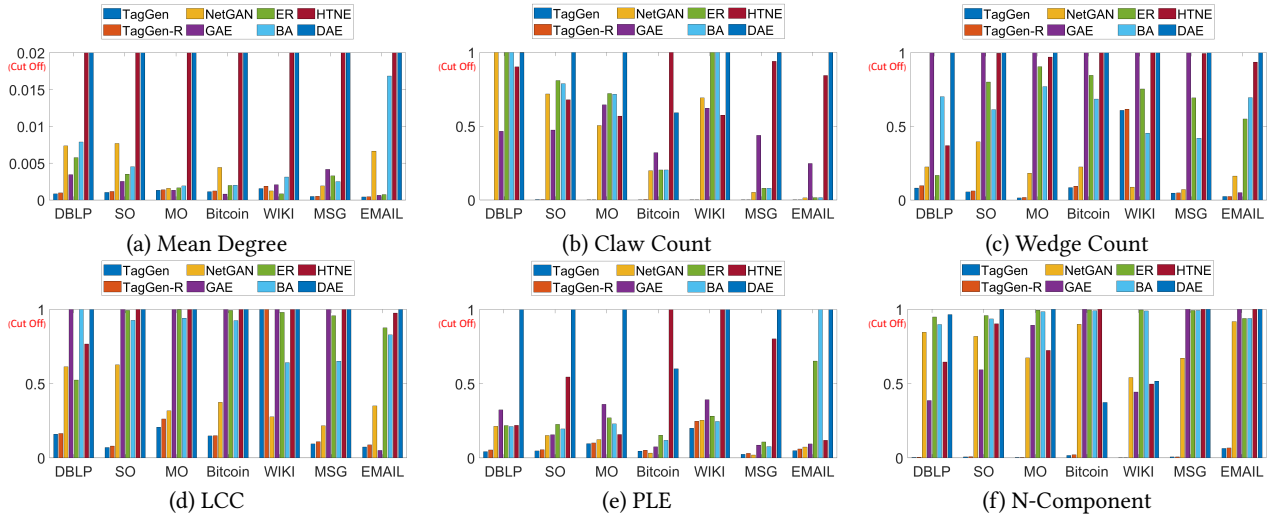


Figure 7: Median score $f_{med}(\cdot)$ comparison on six metrics across seven temporal networks. Best viewed in color. We cut off high values for better visibility. (Smaller metric values indicate better performance)

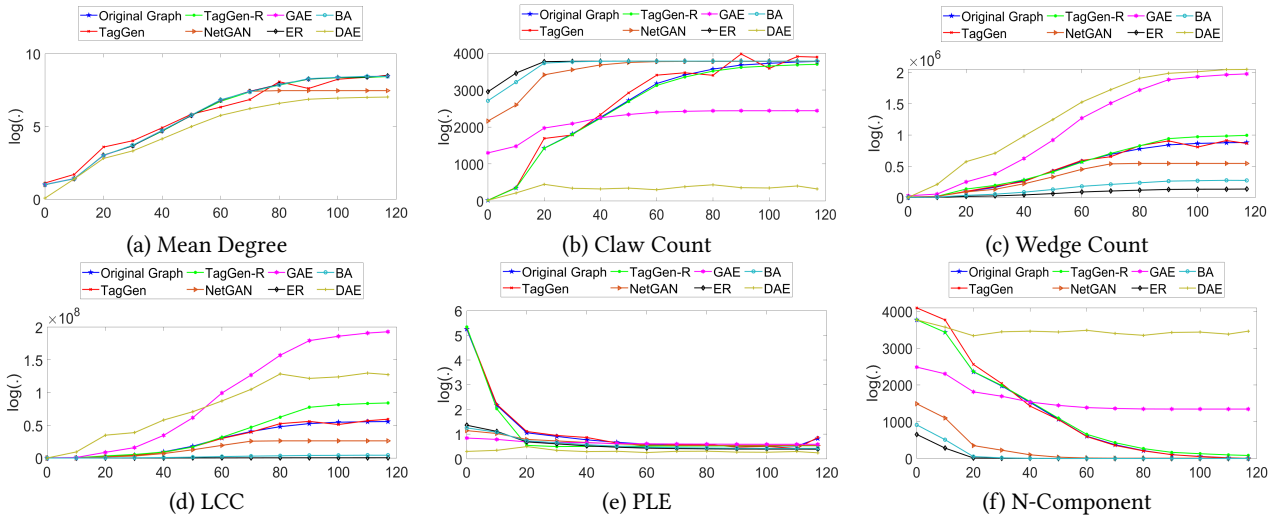


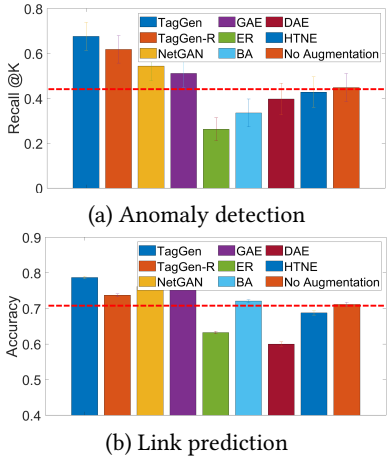
Figure 8: The comparison results on the six evaluation metrics across 117 timestamps in BITCOIN data set. Best viewed in color. The algorithm better fitting the curve of the original graph (colored in blue) is better.

while falling short of capturing many other network properties (e.g., LCC and wedge count).

To further demonstrate the performance of *TagGen*, we experiment with the BITCOIN data set and evaluate the performance of all algorithms in terms of six different metrics in each timestamp. By doing this, we want to explore how the performances of the different methods change over 117 timestamps in the BITCOIN data set. The experimental results are shown in Fig. 8, where the X-axis represents timestamp, and the Y-axis represents the value of a metric (labeled under each figure). In general, we observe (1) all the methods perform similarly well on Mean Degree metric; (2) *TagGen* consistently performs better than the baseline methods across six metrics and 117 timestamps as *TagGen* (colored in red) better fits the curves of the original graph (colored in blue). A simple guess here is that *TagGen* is the only dynamic graph generative model that can better track the trend of network evolution.

4.3 Case Studies in Data Augmentation

Anomaly Node Detection: In real-world networks, the performance of anomaly detection algorithms is often degraded due to data sparsity. Here, we conduct a case study of boosting the performance of anomaly node detection in SO data set via data augmentation. In particular, we select the labeled network SO as our evaluation data and consider a minority class (8%) in SO as the anomalies. In particular, we conduct 10-fold cross-validation and



(a) Anomaly detection

(b) Link prediction

Figure 9: Data Augmentation in SO

employ Recall@K as the evaluation metric, where K is the total number of anomaly nodes in the test set. To assess the performance of anomaly node detection with data augmentation, we use the generative models to synthesize temporal edges and inject them into the original graph. Then, we encode the augmented temporal network into a node-wised representations [27], which is fed into the logistic regression model as inputs for classifying the malicious nodes. The experimental results are shown in Fig. 9 (a), where No Augmentation (red dotted line) shows the result (Recall@K = 44.8%) of logistic regression directly trained on the embedding of the original graph without augmentation. The height of the bars indicates the average value of Recall@K, and the error bars represent the standard deviation in 10 runs. We observe that our proposed method boosts Recall@K to 67.6% (22.8% improvement over the base model No Augmentation), while our best competitor NetGAN only achieves 54.3% (9.5% improvement over No Augmentation).

Link Prediction: In this experiment, we randomly select 50% of edges as the training data and the rest as the test data. Then, we compute the node embedding of both the original graph and the generated graph via CTDNE [27]. At last, we concatenate the two sets of node embedding and feed them into a logistic regression model to perform link prediction on the test data. In Fig. 9 (b), the height of the bars indicates the average value of accuracy, and the error bars represent the standard deviation in 10 runs. It can be seen that NetGAN and GAE barely improve the performance of link prediction, while our proposed method *TagGen* increases the accuracy rate by 2.7% over the base model without data augmentation.

5 RELATED WORK

In this section, we briefly review the related works regarding dynamic network mining and graph generative model.

Dynamic Network Mining. Recently, significant research interests have been observed in developing deep models for dynamic networks. Most existing work models the dynamic networks as time-evolving graphs, which aggregate temporal information into a sequence of snapshots. For instance, [24] proposes a network embedding approach for modeling the linkage evolution in the dynamic network; [26] proposes a graph attention neural mechanism to learn from the spatial-temporal context information of the time-evolving graphs; [41] proposes Spatio-Temporal Graph Convolutional Networks with complete convolutional structures, enabling faster training speed while tackling the issue of the high non-linearity and complexity of traffic flow. However, these approaches may not be able to fully capture the rich temporal context information in the data due to the aggregation over time. For this reason, the authors of [27] proposed to learn network embedding for temporal interaction networks by developing a family of temporally increasing random walks to extract network context information. In this paper, we propose a generic framework to further model and generate the temporal interaction networks by mimicking the network evolution process in real dynamic systems. To the best of our knowledge, *TagGen* is the first deep graph generative model designed for temporal networks.

Graph Generative Model. Early studies of graph generative models include the explicit probabilistic models [8, 9], stochastic block models [11], preferential attachment models [2, 3, 19], exponential random graph models [32], the small-world model [14], and Kronecker graphs [22]. In addition to the static models, some attempts have also been made for generating dynamic graphs. For instances, [10] proposes a dynamic graph generation framework that is able to control the network diameter for a long-time horizon; [31] develops a graph generator that models the temporal motif distribution. However, all of the aforementioned approaches basically generate graphs relying on some prior structural assumptions. Hence, such methods are often hand-engineered and cannot directly learn from the data without prior knowledge or assumptions. The recent progress in deep generative models (e.g., [12, 17]) has attracted a surge of attention to model the graph-structured data. For example, in [5], the authors aim to capture the topology of a graph by learning a distribution over the random walks in an adversarial setting; in [40], the authors propose a framework named Graph-RNN to decompose graph generation into two processes: one is to generate a sequence of nodes, and the other is to generate a sequence of edges for each newly added node. This paper proposes a deep generative framework to model dynamic systems and generate the temporal interaction networks via a family of local operations to perform the addition and deletion of nodes and edges.

6 CONCLUSION

In this paper, we propose *TagGen* - the first attempt to generate temporal networks by directly learning from a collection of timestamped edges. *TagGen* is able to generate graphs that capture important structural and temporal properties of the input data via a novel context sampling strategy together with a bi-level self-attention mechanism. We present comprehensive evaluations of *TagGen* by conducting the quantitative evaluation in temporal graph generation and two case studies of data augmentation in the context of anomaly detection and link prediction. We observe that: (1) *TagGen* consistently outperforms the baseline methods in seven data sets with six metrics; (2) *TagGen* boosts the performance of anomaly detection and link prediction approaches via data augmentation. However, key challenges remain in this space. One possible future direction is to develop generative models that can jointly model

the evolving network structures and node attributes in order to generate attributed networks in the dynamic setting.

ACKNOWLEDGMENTS

This work is supported by National Science Foundation under Grant No. IIS-1618481, IIS-1704532, IIS-1741317, IIS-1947203, and IIS-2002540 the U.S. Department of Homeland Security under Grant Award Number 17STQAC00001-03-03 and Ordering Agreement Number HSHQDC-16-A-B0001, US DARPA KAIROS Program No. FA8750-19-2-1004, SocialSim Program No. W911NF-17-C-0099, a Baidu gift, and IBM-ILLINOIS Center for Cognitive Computing Systems Research (C3SR) - a research collaboration as part of the IBM AI Horizons Network. 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] Steven P. Abney. 2002. Bootstrapping. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*.

[2] Leman Akoglu and Christos Faloutsos. 2009. RTG: a recursive realistic graph generator using random typing. *Data Min. Knowl. Discov.*

[3] Réka Albert and Albert-László Barabási. 2001. Statistical mechanics of complex networks. *CoRR* cond-mat/0106096 (2001).

[4] Yikun Ban, Xin Liu, Ling Huang, Yitao Duan, Xue Liu, and Wei Xu. 2019. No Place to Hide: Catching Fraudulent Entities in Tensors. In *The World Wide Web Conference*.

[5] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. 2018. NetGAN: Generating Graphs via Random Walks. In *Proceedings of the 35th International Conference on Machine Learning*.

[6] Léon Bottou. 2010. Large-Scale Machine Learning with Stochastic Gradient Descent. In *19th International Conference on Computational Statistics, COMPSTAT*.

[7] Dean V Buonomano and Michael M Merzenich. 1995. Temporal Information Transformed into a Spatial Code by a Neural Network with Realistic Properties. *Science* (1995).

[8] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. 2004. R-MAT: A Recursive Model for Graph Mining. In *Proceedings of the Fourth SIAM International Conference on Data Mining*.

[9] Paul Erdös and Alfréd Rényi. 1959. On random graphs, I. *Publicationes Mathematicae (Debrecen)* (1959).

[10] Frank Fischer and Christoph Helmberg. 2014. Dynamic graph generation for the shortest path problem in time expanded networks. *Math. Program.* (2014).

[11] Anna Goldenberg, Alice X. Zheng, Stephen E. Fienberg, and Edoardo M. Airoldi. 2009. A Survey of Statistical Network Models. *Foundations and Trends in Machine Learning* (2009).

[12] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*.

[13] Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. 2020. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning.

[14] Carsten Grabow, Stefan Grosskinsky, Jürgen Kurths, and Marc Timme. 2015. Collective Relaxation Dynamics of Small-World Networks. *CoRR* abs/1507.04624 (2015). arXiv:1507.04624

[15] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[16] Jian Kang and Hanghang Tong. 2019. N2N: Network Derivative Mining. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*.

[17] Diederik P. Kingma and Max Welling. 2014. Auto-Encoding Variational Bayes. (2014).

[18] Thomas N. Kipf and Max Welling. 2016. Variational Graph Auto-Encoders. *CoRR* abs/1611.07308 (2016). arXiv:1611.07308

[19] Jon M. Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. 1999. The Web as a Graph: Measurements, Models, and Methods. In *5th Annual International Conference of Computing and Combinatorics*.

[20] Srijan Kumar, Francesca Spezzano, V. S. Subrahmanian, and Christos Faloutsos. 2016. Edge Weight Prediction in Weighted Signed Networks. In *IEEE 16th International Conference on Data Mining*.

[21] Srijan Kumar, Xikun Zhang, and Jure Leskovec. 2019. Predicting Dynamic Embedding Trajectory in Temporal Interaction Networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[22] Jure Leskovec, Deepayan Chakrabarti, Jon M. Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. 2010. Kronecker Graphs: An Approach to Modeling Networks. *J. Mach. Learn. Res.* (2010).

[23] Jure Leskovec and Andrej Krevl. 2015. {SNAP Datasets}:{Stanford} Large Network Dataset Collection. (2015).

[24] Taisong Li, Jiawei Zhang, Philip S. Yu, Yan Zhang, and Yonghong Yan. 2018. Deep Dynamic Network Embedding for Link Prediction. *IEEE Access* (2018).

[25] Xu Liu, Jingrui He, Sam Duddy, and Liz O'Sullivan. 2019. Convolution-Consistent Collective Matrix Completion. In *International Conference on Information and Knowledge Management*.

[26] Zhining Liu, Dawei Zhou, and Jingrui He. 2019. Towards Explainable Representation of Time-Evolving Graphs via Spatial-Temporal Graph Attention Networks. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*.

[27] Giang Hoang Nguyen, John Boaz Lee, Ryan A. Rossi, Nesreen K. Ahmed, Eunyee Koh, and Sungchul Kim. 2018. Continuous-Time Dynamic Network Embeddings. In *Companion of the The Web Conference 2018 on The Web Conference 2018*.

[28] Pietro Panzarasa, Tore Opsahl, and Kathleen M. Carley. 2009. Patterns and dynamics of users' behavior and interaction: Network analysis of an online community. *J. Assoc. Inf. Sci. Technol.* (2009).

[29] Ashwin Paranjape, Austin R. Benson, and Jure Leskovec. 2017. Motifs in Temporal Networks. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*.

[30] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: online learning of social representations. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[31] Sumit Purohit, Lawrence B Holder, and George Chin. 2018. Temporal Graph Generation Based on a Distribution of Temporal Motifs. In *Proceedings of the 14th International Workshop on Mining and Learning with Graphs*.

[32] Garry Robins, Pip Pattison, Yuval Kalish, and Dean Lusher. 2007. An introduction to exponential random graph (p^*) models for social networks. *Soc. Networks* (2007).

[33] Huajie Shao, Dachun Sun, Jiahao Wu, Zecheng Zhang, Aston Zhang, Shuochao Yao, Shengzhong Liu, Tianshi Wang, Chao Zhang, and Tarek F. Abdelzaher. 2020. paper2repo: GitHub Repository Recommendation for Academic Papers. In *The Web Conference*.

[34] Huajie Shao, Shuochao Yao, Yiran Zhao, Chao Zhang, Jinda Han, Lance M. Kaplan, Lu Su, and Tarek F. Abdelzaher. 2018. A Constrained Maximum Likelihood Estimator for Unguided Social Sensing. In *IEEE Conference on Computer Communications*.

[35] George R Terrell and David W Scott. 1992. Variable Kernel Density Estimation. *The Annals of Statistics* (1992).

[36] Hanghang Tong, Spiros Papadimitriou, Philip S. Yu, and Christos Faloutsos. 2008. Proximity Tracking on Time-Evolving Bipartite Graphs. In *Proceedings of the SIAM International Conference on Data Mining*.

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*.

[38] Bernard M. Waxman. 1988. Routing of multipoint connections. *IEEE J. Sel. Areas Commun.* (1988).

[39] Jiaxuan You, Bowen Liu, Zhitao Ying, Vijay S. Pande, and Jure Leskovec. 2018. Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation. In *Advances in Neural Information Processing Systems*.

[40] Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. 2018. GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models. In *Proceedings of the 35th International Conference on Machine Learning*.

[41] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2018. Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting. (2018).

[42] Si Zhang, Dawei Zhou, Mehmet Yigit Yildirim, Scott Alcorn, Jingrui He, Hasan Davulcu, and Hanghang Tong. 2017. HiDDen: Hierarchical Dense Subgraph Detection with Application to Financial Fraud Detection. In *Proceedings of the 2017 SIAM International Conference on Data Mining*.

[43] Dawei Zhou, Jingrui He, Hongxia Yang, and Wei Fan. 2018. SPARC: Self-Paced Network Representation for Few-Shot Rare Category Characterization. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[44] Dawei Zhou, Kangyang Wang, Nan Cao, and Jingrui He. 2015. Rare Category Detection on Time-Evolving Graphs. In *IEEE International Conference on Data Mining*.

[45] Yuan Zuo, Guannan Liu, Hao Lin, Jia Guo, Xiaoqian Hu, and Junjie Wu. 2018. Embedding Temporal Network via Neighborhood Formation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

A NOTATIONS

Symbol	Description
$\widetilde{G} = (\widetilde{V}, \widetilde{E})$	temporal interaction network
$\widetilde{V} = \{v_1, \ldots, v_n\}$	the set of nodes in $\widetilde{G}$
$\widetilde{E} = \{e_1^{te_1}, ..., e_m^{te_m}\}$	the set of temporal edges in $\widetilde{G}$
$v = \{v^{t_1}, \ldots, v^{t_T}\}$	node v with its occurrences at $\{t_1, \ldots, t_T\}$
$e^t = (u, v)^t$	temporal edge between u and v at timestamp t
$W = \{w_1, \ldots, w_k\}$	the k-length temporal walk
$\widetilde{W} = \{\widetilde{w}_1, \ldots, \widetilde{w}_k\}$	the synthetic k-length temporal walk
$\mathcal{N}_{FT}(\cdot)$	the neighborhood function
n	the total number of nodes
n_r	the total number of temporal occurrences
m	the total number of temporal edges
T	the total number of timestamps in $\widetilde{G}$
$\odot$	Hadamard product

Table 1: Symbols

B ALGORITHM ANALYSIS

Lemma 1. *For any $v^{t_v} \in \widetilde{V}$, if the temporal neighborhood distribution $p(v^{t_v}|\mathcal{N}_T(v^{t_v}))$ and topology neighborhood distribution $p(v^{t_v}|\mathcal{N}_S(v^{t_v}))$ are weakly dependent on each other, then the following inequality holds:*

$$p(v^{t_v}|\mathcal{N}_{FT}(v^{t_v})) \geq \alpha \frac{p(v^{t_v}|\mathcal{N}_S(v^{t_v}))p(v^{t_v}|\mathcal{N}_T(v^{t_v}))p(\mathcal{N}_S(v^{t_v}))p(\mathcal{N}_T(v^{t_v}))}{p(\mathcal{N}_S(v^{t_v}), \mathcal{N}_T(v^{t_v}))} \tag{8}$$

where $\alpha = \frac{\delta}{p(v^{t_v})}$.

Proof. For any $v^{t_v} \in \widetilde{G}$, the context importance $p(v^{t_v}|\mathcal{N}_{FT}(v^{t_v}))$ can estimated as

$$\begin{aligned} p(v^{t_v}|\mathcal{N}_{FT}(v^{t_v})) &= p(v^{t_v}|\mathcal{N}_S(v^{t_v}), \mathcal{N}_T(v^{t_v})) \\ &= \frac{p(v^{t_v}, \mathcal{N}_S(v^{t_v}), \mathcal{N}_T(v^{t_v}))}{p(\mathcal{N}_S(v^{t_v}), \mathcal{N}_T(v^{t_v}))} \end{aligned} \tag{9}$$

Since the corresponding temporal neighborhood distribution $p(v^{t_v}|\mathcal{N}_T(v^{t_v}))$ and topology neighborhood distribution $p(v^{t_v}|\mathcal{N}_S(v^{t_v}))$ satisfy a weak dependence, we can easily have

$$\begin{aligned} & p(v^{t_v}|\mathcal{N}_{FT}(v^{t_v})) \\ & \geq \delta \frac{p(v^{t_v})p(\mathcal{N}_S(v^{t_v})|v^{t_v})p(\mathcal{N}_T(v^{t_v})|v^{t_v})}{p(\mathcal{N}_S(v^{t_v}), \mathcal{N}_T(v^{t_v}))} \\ & = \delta \frac{p(v^{t_v}) \frac{p(v^{t_v}|\mathcal{N}_S(v^{t_v}))p(\mathcal{N}_S(v^{t_v}))}{p(v^{t_v})} \frac{p(v^{t_v}|\mathcal{N}_T(v^{t_v}))p(\mathcal{N}_T(v^{t_v}))}{p(v^{t_v})}}{p(\mathcal{N}_S(v^{t_v}), \mathcal{N}_T(v^{t_v}))} \\ & = \alpha \frac{p(v^{t_v}|\mathcal{N}_S(v^{t_v}))p(v^{t_v}|\mathcal{N}_T(v^{t_v}))p(\mathcal{N}_S(v^{t_v}))p(\mathcal{N}_T(v^{t_v}))}{p(\mathcal{N}_S(v^{t_v}), \mathcal{N}_T(v^{t_v}))} \end{aligned} \tag{10}$$

□

C DATA STATISTICS

We evaluate *TagGen* on seven real temporal networks. Specifically, DBLP [43] is a citation network that contains bibliographic information of the publications in IEEE Visualization Conference from 1990 to 2015; SO [43] and MO [29] are two collaboration networks where each node represents a user, and the edge represents one user's comments on another user; WIKI [23] is a voting network, where each edge exists if the contributors vote to elect the administrators; EMAIL [29] and MSG [28] are two communication networks, where an edge exists if one person sends at least one email/message to another person at a certain timestamp; BITCOIN [20] is a who-trusts-whom network where people trade with bitcoins on a Bitcoin Alpha platform.

Network	Nodes	Temporal Edges	Timestamps
EMAIL	986	332,334	26
DBLP	1,909	8,237	15
WIKI	7,118	95,333	6
MSG	1,899	20,296	28
BITCOIN	3,783	24,186	117
SO	3,262	13,077	36
MO	13,840	195,330	20

Table 2: Statistics of the network data sets.

D IMPLEMENTATION DETAILS

In the experiments, we set the batch size to be 128, the number of epochs to be 30, the representation size of the node embedding to be 80, the number of head to be 4, the initial learning rate to be 0.003, the bi-level self-attention parameter $\lambda = 0.5$, $d_{\mathcal{N}_{FT}} = 1$, $t_{\mathcal{N}_{FT}} = 1$, the number of initial nodes l to be the number of the total nodes, walks per initial nodes $\gamma = 3$, walk length $k = 20$, and constants $c_1 = 20$ and $\xi = 0.6$. Besides, we outline the computation formula and description regarding the six evaluation metrics used in our experiments in Table 3. All the code and data are publicly available at an anonymous Github repository*. The experiments are performed on a Windows machine with eight 3.8GHz Intel Cores and a single 16GB RTX 5000 GPU.

E ADDITIONAL RESULTS

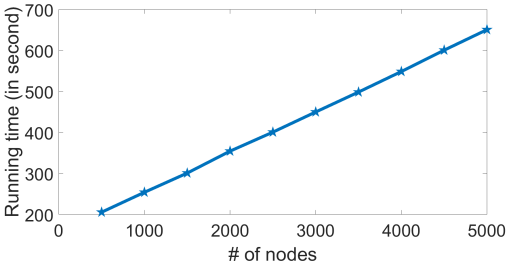
We analyze the scalability of *TagGen*, by recording the running time (i.e., the sum of the training time and the time for graph generation) of *TagGen* on a series of synthetic graphs with increasing size. To be specific, we generate the synthetic graphs via ER algorithm [9], by which we can easily control the number of nodes and the number of edges in a graph. In the experiments, we set the batch size to be 128, the length of the random walk to be 20, the number of epochs to be 30, i.e., the same parameter settings as in the previous subsection. In Fig. 10 (a), we fix the edge density to be 0.005, set the initial number of nodes to be 500, and increase the number of nodes by 500 each time. In Fig. 10 (b), we fix the number of nodes to be 5,000 and increase the edge density from 0.005 to 0.05. Based on the

*https://github.com/davidchouzdw/TagGen

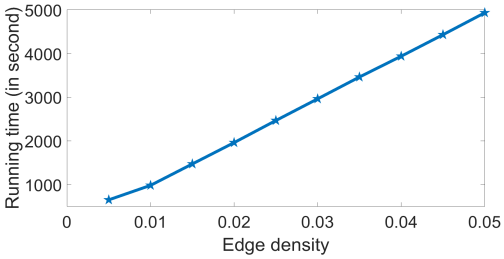
results in Fig. 10, we observe that the complexity of the proposed method is almost linear to the number of nodes. Besides, when we fix the number of nodes and increase the edge density, the running time also increases linearly.

Metric name	Computation	Description
Mean Degree	$\mathbb{E}[d(v)]$	Mean degree of nodes in the graph.
Claw Count	$\sum_{v \in V} \binom{d(v)}{3}$	Number of the claw of the graph.
Wedge Count	$\sum_{v \in V} \binom{d(v)}{2}$	Number of wedges of the graph.
LCC	$\max_{f \in F} \|f\|$	Size of the largest connected component of the graph, where F is the set of all connected components in the graph.
PLE	$1 + n(\sum_{u \in V} \log(\frac{d(u)}{d_{min}}))^{-1}$	Exponent of the power-law distribution of the graph.
N-Component	$\|F\|$	Number of connected components, where F is the set of all connected components in the graph.

Table 3: Graph statistics for measuring network properties.



(a) Running time vs. # of nodes



(b) Running time vs. edge density

Figure 10: Scalability Analysis