TG-GAN: Continuous-time Temporal Graph Deep Generative Models with Time-Validity Constraints

ABSTRACT

Deep generative models of graph-structured data have been experiencing fast developments in very recent years, which majorly focus on static graphs in applications such as molecular design and social networks. However, there are many real-world problems that involve temporal graphs whose topology and attribute values evolve dynamically over time. Sophisticated and unknown network processes have been involved in temporal graphs that cannot be comprehensively assumed by prescribed models in crucial applications such as social mobility networks and cybersecurity catastrophic failures. These web-scale applications challenged current deep graph generative models significantly due to the unique difficulties in achieving: 1) time-validity constraints. 2) time and topological distributions, and 3) jointly temporal and graph encoding and decoding. To address them, this work proposes the "Temporal Graph Generative Adversarial Network" (TG-GAN) model for continuous-time graph generation with time-validity constraints. It can jointly generate the time, node, and edge information for truncated temporal walks via a novel recurrent-based model and a valid time decoder. The generated truncated temporal walks are then assembled to time-budgeted temporal walks for temporal graphs under the learned topological and temporal dependencies. In addition, a temporal graph discriminator is proposed that combines time and node encoding operations over a recurrent architecture to distinguish generated sequences from real ones sampled by a truncated temporal walk sampler. Extensive experiments on both synthetic and real-world datasets confirm that TG-GAN significantly outperforms five bench-marking methods in terms of efficiency and effectiveness.

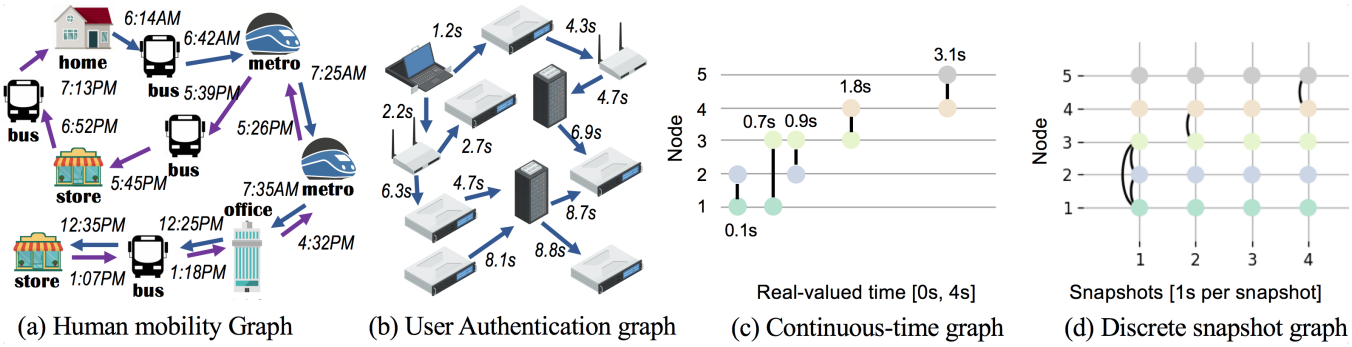
1 INTRODUCTION

Generative models for graphs are powerful approaches that can leverage vast quantities of unannotated graph data to synthesize unknown graph data distributions and fuel subsequent graph data mining tasks. Deep generative models for graphs such as GraphRNN [32], NetGAN [4], and GraphVAE [25] have started to achieve significant success compared to traditional *prescribed* approaches from random graph theory [8, 10]. These deep generative models are applied to static graphs such as molecule generation [25], social communities [32], and citation networks [4], benefiting from their high expressiveness in learning underlying complex principles directly from the data in an end-to-end fashion without the need for handcrafted rules.

However, many real-world graphs are actually *temporal graphs* that are networks evolving dynamically over time. Figure 1 (a) illustrates a typical example of an urban mobility graph for a person, and Figure 1 (b) shows a user authentication graph that reflects the behavior of a computer network administrator who visits from devices to devices. Further application areas include network behavior synthesis and intervention, brain network simulation, and protein folding [14]. Given the underlying complex but unknown network processes involved in many such applications, this paper focuses on highly expressive deep models that can distill and model the underlying generative process of such temporal graphs. This approach goes beyond related domains such as representation learning for dynamic graphs [12] and temporal link prediction [21, 30].

So far, deep generative models for temporal graphs have not been well explored, due to the technical challenges involved in leveraging static graph techniques: **1) Difficulties in ensuring the temporal validity of the generated graphs.** The connectivity patterns for temporal graphs can be very different from those in static graphs. For example, in a temporal graph, if we want to pass a message from node a to node c via node b, we need to know not only the existence of the edges from a to b and b to c, but also that the edge from b to c cannot disappear before the birth of the edge from a to b. There are many such types of temporal validity requirements for temporal graphs, none of which can be considered or ensured using current static graph generation techniques. **2) Challenges in learning temporal graph distributions in both the discrete-valued topological and continuous-valued time dimensions** Temporal edges only exist at a certain time span. Modeling the generative models of temporal graphs requires to characterize the distributions of the time spans and topological dependencies among temporal edges. Conventional formulation of temporal graphs into discrete snapshots of adjacency matrices [12] or individual edges [33] without the characterization of time distributions incur information loss and low efficiency. The former is caused due to the discretization of time into ordinal values while the latter is incurred by highly sparse representation in the joint topological and temporal high dimensional space. **3) Difficulties in jointly encoding and decoding topological and temporal information.** The node and edge patterns on temporal graphs are simultaneously influenced by the graphs' topology and temporal dependency, which also mutually impact each other. For example, the same path in a temporal graph may have a different meaning if it happens during a different time frame.

To address these challenges, we propose a new model, called "Temporal Graph Generative Adversarial Network" (TG-GAN) for continuous-time temporal graph generation with time-validity constraints. It can generate the time, node, and edge information for time-budgeted temporal walks, which are then assembled to temporal graphs under the learned topological and temporal dependencies. The proposed method is very efficient in generatively modeling non-small temporal graphs in the dimensions of both topology and time. We propose a novel temporal graph generator that jointly models truncated temporal walks via a novel recurrent-structured model that enforces temporal validity constraints. We also propose a new encoder and decoder for modeling continuous timestamps and discrete nodes in edges. The major contributions of this paper can be summarized as follows:



(a) Human mobility Graph (b) User Authentication graph (c) Continuous-time graph (d) Discrete snapshot graph

Figure 1: Real-world temporal graphs: *(a)* A transportation graph capturing passenger movement as trips (arrows) between locations (nodes) at given times; *(b)* A user authentication graph to model cyber behavior - a laptop connecting to different desktops, gate servers, production servers, and data servers chronologically. IP addresses are nodes, and remote connections are edges. Two types of temporal graph representations include: *(c)* a continuous-time graph representation has no information loss since the activation time of an edge is a real value ranging from 0 to $4s$; and *(d)* a discrete-snapshot graph with information loss since the activation times are cast to integer values and loose their chronological order. For example, edges $(1, 2, 0.1s)$, $(1, 3, 0.7s)$, $(2, 3, 0.9s)$ can be ordered in continuous-time temporal graph, however, they are converted to new edges of $(1, 2)$, $(1, 3)$, $(2, 3)$ within a snapshot without chronological order.

- **Propose a novel deep generative framework for continuous-time temporal graph generation**. The proposed framework can efficiently and effectively learn the underlying distribution of continuous-time temporal graphs. It generates graphs with time-varying node features while ensuring temporal validity.
- **Develop a new truncated temporal walk generator**. Encoders and decoders for both continuous-valued time information and discrete-valued node information have been proposed. New time budgeting strategies and activation functions have been proposed to ensure temporal validity.
- **Propose a new truncated temporal graph discriminator.** A recurrent-architecture-based discriminator is developed to jointly examine the sequence patterns and time validity along with a truncated temporal walk sampler to obtain real samples from observed temporal graphs.
- **Conduct extensive experiments on both synthetic and real-world data.** The results demonstrate that TG-GAN is capable of generating temporal graphs closely resembling real graphs. It significantly outperforms other models in several metrics by two orders of magnitude with outstanding scalability.

2 RELATED WORK

Temporal graph generation Conventional methods are based on *prescribed* structural assumptions, including probabilistic models [1], configuration models [2], and stochastic block models (SBM) [29]. These prescribed approaches capture some of the predefined properties of a graph, such as the degree distribution, community structure, or clustering patterns. Relevant extensions for temporal graphs are based on these prescribed models and also include models with new designs, such as Stochastic block transition models [28, 31] using a Hidden Markov Model along SBM or Temporal Stochastic Block Model (TSBM) [7]. However, some recent works have identified surprising properties of real-world large graphs and demonstrate that prescribed models are insufficient. More details can be found in surveys such as [16, 23].

Deep generative models for graphs Deep graph generation is based on the concept of different unsupervised learning techniques and includes approaches such as adversarial networks (NetGAN [4]), variational autoencoders (GraphVAE [25]), and recurrent networks (GraphRNN [32]). GraphVAE [25] is a new and first-of-its-kind variational autoencoder for whole graph generation, though it typically only handles very small graphs and does not scale well to large graphs because of both memory and runtime complexity. GraphRNN [32] models a graph as a sequence of node and edge generation that can be learned by autoregressive models, achieving a much better performance and scalability than GraphVAE. NetGAN [4] follows a GAN approach [11] and generates synthetic random walks while discriminating synthetic walks from real random walks sampled from a real graph. These approaches typically generate a synthetic static graph of good quality. Although these popular generative approaches for many applications, only one very recent approach discusses a deep model for temporal graph generation. TagGen [33] models deep generative processes of k-length temporal walks to compose a temporal interaction networks, which is a specific type of general temporal graphs that reduces time information into ordinal values. Hence it does not handle continuous-time temporal graphs, time validity constraints, and time distributions. Given all the above work, it is imperative to propose deep generative models for learning the distributions of generic temporal graphs in both graph topology and continuous-time dimensions with temporal validity guarantee.

Random walk For static graphs, random walks are shown to be a powerful representation when it comes to scalability [4]. The random walk is a diffusion model and its dynamicity provides fundamental hints of the underlying mechanisms for a whole class of diffusive processes in graphs. Temporal graphs basically depict time-dependent and time-constrained diffusion, so temporal walks,

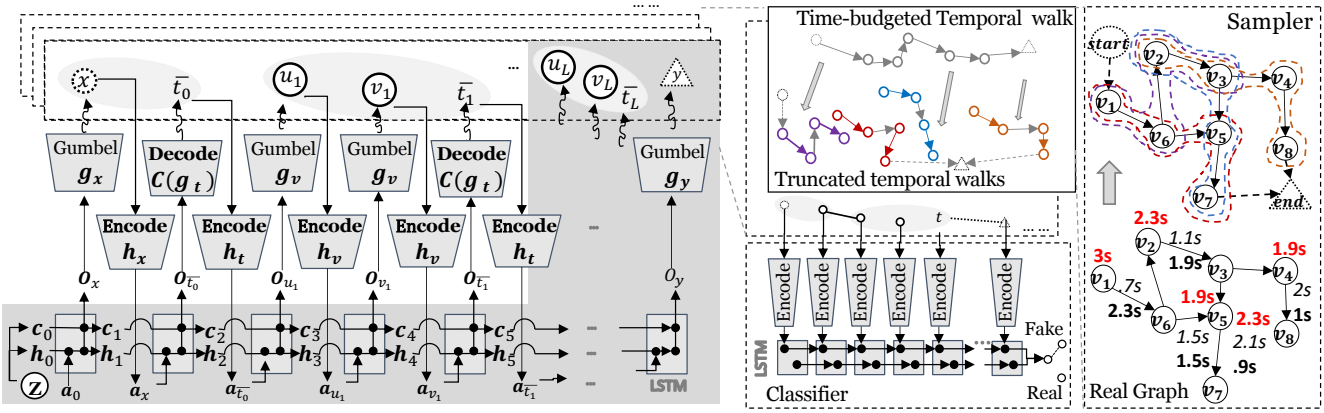


Figure 2: The outline of the TG-GAN framework.

extension of random walks in temporal graphs, inherit their diffusion properties [15] with additional time-dependent characteristics. For example, continuous-time networks are sequentially activated by edge connections within start and end times [21], hence, to activate certain sub-graphs therefore be invalid as time evolves. They are already popular in representation learning approaches, such as DeepWalk [22], dynamic network embeddings [21], and dynamic representation learning [24]. To the best of our knowledge, TagGen [33] is the only existing work attempting to learn the latent generative distribution for temporal interaction networks via time-budgeted temporal walks. In this work, we propose a powerful model for a wider range of continuous-time temporally-bounded dynamic graphs.

3 DEEP GENERATIVE MODELS FOR TEMPORAL GRAPH GENERATION

This section focuses on formulating the problem and then describing our proposed TG-GAN approach, which 1) directly models *time-dependent* and *time-constrained* diffusion dynamics in varied-length sequences, 2) accomplishes graph generation through sequential models that do not have a permutation requirement, and 3) possesses a time complexity independent of the number of nodes.

3.1 Problem Formulation

A temporal graph is a directed graph $G = \{e_1, e_2, \cdots, e_M\}$, $e_i = (u_i, v_i, t_i)$ is a temporal edge, where $u_i, v_i \in V$ are respectively start and end nodes from node set V, $t_i \in T$ is the time point of the edge, $T = [0, t_{end}]$ (where $t_{end} \subseteq \mathbb{R}^+$) is the time span of the temporal graph. A *temporal walk* is defined as $s = \{e_1, e_2, \cdots, e_{L_s}\} = \{(u_1, v_1, t_1), \cdots, (u_{L_s}, v_{L_s}, t_{L_s})\}$, where $\forall e_i \in G$ and L_s is the length of the walk s. Hence s is terminated at the edge $e_{L_s} = (u_{L_s}, v_{L_s}, t_{L_s})$ whose end node v_{L_s}'s neighbor edges are all earlier than t_{L_s}. Therefore, a temporal graph can be also denoted as the union of all the temporal walks in it $G = \bigcup_{s \sim p_G(s)} s$, where $p_G(s)$ is the distribution of all the walks in graph G. The central problem of deep generative modeling for temporal graphs is to learn an underlying distribution $G \sim p(G)$ for a set of temporal graphs, each of which can be represented by a set of temporal walks.

3.2 Overall Architecture

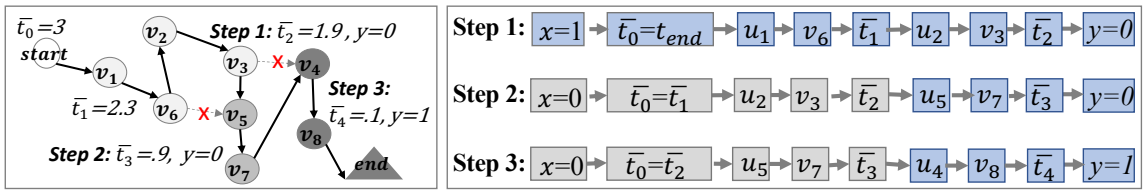
TG-GAN captures the topological and temporal patterns of temporal graphs by learning the distributions of temporal walks. It consists of two parts, namely a temporal walk generator $\mathcal{G}$ (a generative recurrent model, shown in the lower part of Figure 2 (a)) and a temporal walk discriminator $\mathcal{D}$ (another recurrent model, shown in the right lower part of Figure 2 (b)) in a MinMax game. The key challenge here is that the generated temporal walk $\mathcal{G}(\boldsymbol{z})$ must inherit strong temporal diffusion and satisfy the temporal constraint, whose set of temporal-valid solutions is defined as C. Overall, the objective function of TG-GAN with temporal constraints is:

$$\min_{\mathcal{G}} \max_{\mathcal{D}} \left[\mathbb{E}_{G, s \sim p_G(s)}[log(\mathcal{D}(s))] + \mathbb{E}_{\boldsymbol{z} \sim p(\boldsymbol{z})}[log(1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z})))]\right]$$
$$s.t.\ \mathcal{G}(\boldsymbol{z}) \in \mathbb{C} \quad (1)$$

where s is a real temporal walk sampled from observed temporal graphs, and $\boldsymbol{z} \sim p(\boldsymbol{z})$ follows a trivial distribution such as an isotropic Gaussian. $\mathbb{C} = \{s | \mathcal{T}(s) \in \mathbb{T}, s \sim p_G(s), G \sim p(G)\}$ is the set of all the truncated temporal walks whose edges' times $\mathcal{T}(s)$ follow the required temporal validity constraint $\mathbb{T}$. This will be discussed in Section 3.6. The general ideas of the generator and discriminator are as follows: **1) The generator** $\mathcal{G}$, illustrated in Figure 2, trains a fixed-length LSTM whose output $\mathcal{G}(\boldsymbol{z})$ is a sequence of special temporal walks defined in the next Section 3.3 with a novel assembly method (as detailed in Section 3.4) and a non-parametric time distribution inference (as detailed in Section 3.5). Time constraint $\mathcal{G}(\boldsymbol{z}) \in \mathbb{C}$ is handled with a new time constraint operation (as detailed in 3.6). **2) the discriminator** $\mathcal{D}$, illustrated in Figure 2 (b), tried its best to score real and fake temporal walks so that expectation of loss $1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z}))$ for discriminating generated samples is minimized while expectation of loss $\mathcal{D}(s)$ for discriminating real samples is maximized. Its details and an underlying classifier design is introduced in Section 3.7. The final output consists of the classifying scores in the GAN framework.

3.3 Generation via truncated temporal walks with time budgets

Unlike static connected graphs where random walks can have fixed length and hence easy to learn, walks in temporal graphs come with



(a) A whole time-budgeted temporal walk (b) Truncated temporal walks and assemble them incrementally

Figure 3: (a) Example of a time-budgeted temporal walk. Shades of grey indicate nodes of the same step. Two *red crosses* mark two impossible edges in this walk; (b) for training, three truncated temporal walks are used as real walks by the discriminator. For inference, truncated temporal walks are assembled step-by-step (blue squares) to ensure strong temporal diffusion. *Step 1)*: generate two temporal edges (blue squares) and $y = 0$ with $x = 1, \bar{t}_0 = t_{end}$ as inputs; *Step 2)*: generate one additional edge and $y = 0$ profile with inputs of last edge $(u_2, v_3, \bar{t}_2)$ and profile of $x = 0, \bar{t}_1$; *Step 3)*: generate another edge and $y = 1$ with inputs of last edge $(u_5, v_7, \bar{t}_3)$ and profile of $x = 0, \bar{t}_2$, and the generation terminates at $y = 1$.

starting point and end point and hence their lengths can vary significantly. For example, for a temporal graph with 1 million nodes, the length could range from 1 to 1 million which is extremely difficult for typical sequential models to learn. To solve this issue, one approach is to utilize fixed-length sequential models. However, naively splitting a temporal walk s into smaller sized walks eliminates the temporal dependence at the split points. We thus need a way to connect these truncated walks with their preceding and succeeding walks, leading to the following definitions:

Definition 1 (Time-budgeted Temporal Walks). *A **time-budgeted temporal walk** is defined as $s = \{\bar{t}_0, \bar{e}_1, \bar{e}_2, \cdots, \bar{e}_{L_s}, \bar{t}_{end}\}$, where L_s is the length of this walk. $\bar{e}_i$ are time-inversed edges as $\bar{e}_i = (u_i, v_i, \bar{t}_i) \in E$, where $\bar{t}_i = t_{end} - t_i$ is called the* "time budget" *for the edge $\bar{e}_i$ and consequently $\bar{t}_0 = t_{end}$ and $\bar{t}_{end} = 0$, which denote the respective full time budget ($= t_{end}$) and end time budget ($= 0$).*

Definition 2 (Truncated Temporal Walks). *A **truncated temporal walk** is defined as a sequence $\bar{s} = \{c, \bar{e}_1, \bar{e}_2, \cdots, \bar{e}_L\}$ with its profile information $c = (x, y, \bar{t}_0)$, which is truncated from an originated* time-budgeted temporal walk. *The temporal edges $\bar{e}_i, i = 1, \cdots, L$, where L is equal to or less than a threshold defined as the maximal length of a truncated temporal walk. Here, the profile $c = (x, y, \bar{t}_0)$ includes $x \in \{0, 1\}$ and $y \in \{0, 1\}$, which denote whether $\bar{s}$ is the respective starting or ending point ($= 1$) or neither of them ($= 0$) in its originated* time-budgeted temporal walk.

For example, in Figure 3 (a), a time-budgeted temporal walk with a length of 4 has been concatenated as 3 truncated temporal walks with fixed length of 2, where the first one starting at u_1 has a profile time budget of $\bar{t}_0 = t_{end} = 3s$ ("s" here means a second here) and its starting flag is $x = 1$. Raw time stamps are shown in italics. The temporal walk has two edges, namely $(u_1, v_6, 2.3s)$ and $(u_2, v_3, 1.9s)$. Since it is not the end of the time-budgeted temporal walk, its end flag is $y = 0$. $2.3s$ and $1.9s$ represent the remaining time budgets after both edges have been created at times $0.7s$ and $1.1s$. The second truncated temporal walk starts at $(u_2, v_3, 1.9s)$ with a profile of $x = 0$, and $\bar{t}_0 = 2.3s$, where $2.3s$ comes from the previous edge $(u_1, v_6, 2.3s)$. The second edge is $(u_5, v_7, 0.9s)$, and finally, $y = 1$ for this walk. Similarly, the third truncated temporal walk includes edges $(u_5, v_7, 0.9s)$ and $(u_4, v_8, 0.1s)$ with $y = 1$. The time-budgeted also terminates here since $y = 1$. The v_6, v_5 edge and v_3, v_4 edge are red-crossed because there is not enough time budget for these two edges.

The generator generates truncated temporal walks $\bar{s}$ during the training phase and assemble time-budgeted temporal walks s in the inference phase as follows:

Training phase: The overall architecture used by the generator is illustrated in Figure 2 during training phase to generate $\bar{s}$. Specifically, the backbone structure of generator is a fixed-length LSTM. The basic LSTM unit has latent state $\boldsymbol{h}_i$ and cell state $\boldsymbol{c}_i$. $\boldsymbol{a}_i$ is an input, and $\boldsymbol{o}_i$ is an output. The generator outputs the generated time budgets $\bar{t}_i$ and nodes u_i, v_i through decoding $\boldsymbol{o}_i$, according to different categorical and continuous data decoding modules. The categorical data decoding modules, which include: node v_i decoding function g_v so that $v_i = g_v(\boldsymbol{o}_{v_i})$; the start profile x decoding function g_x so that $x = g_x(\boldsymbol{o}_x)$; and the end profile y decoding function g_y so that $y = g_y(\boldsymbol{o}_y)$. The continuous data decoding module includes a time distribution inference module $g_t(\cdot)$ and its auxiliary time-constraint functions C so that $\bar{t}_i = C(g_t(o_{\bar{t}_i}))$. The generator also uses different encoding modules to encode generated data to the inputs $\boldsymbol{a}_i$. There are categorical encoding modules, including: the start profile x encoding function h_x so that $\boldsymbol{a}_x = h_x(x)$; and nodes v_i encoding function h_v so that $\boldsymbol{a}_{v_i} = h_v(v_i)$. The continuous data encoding module is just a function h_t so that $\boldsymbol{a}_{\bar{t}_i} = h_x(\bar{t}_i)$. The categorical data decoding functions utilize a Gumbel-max re-parameterization (as detailed in Appendix A.1 and A.2). Its novel decoding functions for continuous-time distribution is described in Sections 3.5 and 3.6.

Inference phase: the generator assembles (varying-length) a time-budgeted temporal walk s by stacking edges step-by-step illustrated in Figure 3 (b). The first truncated temporal walk $\bar{s} = \{x = 1, y = 0, \bar{t}_0 = t_{end}, (u_1, v_6, \bar{t}_1), (u_2, v_3, \bar{t}_2)\}$ is generated by LSTM with the only input of random variable $\boldsymbol{z}$. The next step recycles $\bar{t}_0 = \bar{t}_1$ and $(u_2, v_3, \bar{t}_2)$ and generates only one more new edge $(u_5, v_7, \bar{t}_3)$. Then, the next step recycles $\bar{t}_0 = \bar{t}_2$ and $(u_5, v_7, \bar{t}_3)$, and generates another new edge $(u_4, v_8, \bar{t}_4)$ with $y = 1$. These generation steps are terminated now since $y = 1$ is found. The details of the proposed inference method are described in Section 3.4.

3.4 Time-budgeted temporal walk assembly using the truncated temporal walk generator

After the generator has been trained, we can compose time-budgeted temporal walks from a set of generated truncated walks. However, arbitrarily matching and chronologically concatenating them will not be sufficient, since this approach does not preserve the temporal dependency of two consecutive truncated walks. Instead, we strive to take into account temporal dependency by the following two types of underlying diffusion: *1) Weak temporal diffusion.* This assumes a non-Markovian process for the whole temporal graph. Under this assumption, one truncated walk can be freely assembled with any other truncated temporal walk under only a loose chronological requirement of $t_i \leq t_j$; and *2) Strong temporal diffusion.* This assumes that the connection of two consecutive edges follows a Markovian process. This means that chronologically assembling different walks is only a necessary, but not a sufficient condition to connect two truncated temporal walks. Strong temporal diffusion is a stricter approach and existing works on temporal graphs [21, 24, 33] only consider the weak temporal diffusion, here we model the stricter strong temporal diffusion case.

To guarantee strong temporal diffusion, when generating each temporal edge $\bar{e}_i$, we need to maintain its conditional dependency $p(\bar{e}_i, y_0 | \bar{e}_{i-1}, \bar{e}_{i-2}, ...\bar{e}_1, x_0, t_0)$ over all the historical edges in a time-budgeted temporal walk. As shown in Figure 3 (b), this is done by first generating an initial truncated temporal walk (all blue squares) in Step 1 and then incrementally appending additional temporal edges $\bar{e}_i$ in Step 2 and 3 (blues squares after grey squares recycled from last step), one at a time, finishing until an end status flag is $y = 1$ in Step 3. All generating processes re-use the same generator. Such inference model directly the strong temporal diffusion in temporal graphs.

3.5 End-to-end time distribution inference

The time budget $\bar{t}_i$ of each temporal edge is assumed to be sampled from the underlying distribution handled by a time distribution inference module g_t and an auxiliary time-constraint module C so that $\bar{t}_i = C(g_t(o_{\bar{t}_i}))$. In this part, we describe g_t first. A conventional way to achieve this is to assume a prescribed distribution, e.g., Gaussian, Gamma, or Beta distribution, followed by re-parameterization tricks to move the non-differentiable sampling operations from the sufficient statistics parameters (e.g., the mean and variance of the Gaussian) to a unit Gaussian $\mathcal{N}(0, 1)$, as was done for the variational decoder in [19]. Unfortunately, the time budget in many real-world situations does not follow for a simple distribution and the true distribution is typically unknown, or cannot be described by known prescribed distributions. It is thus highly desirable for the model to identify and fit such unknown distributions with highly expressive generative models.

Here, we propose a deep end-to-end time distribution inference method, namely "**Time Decoder**" which is illustrated in Figure 4. The time decoder is a end-to-end sampling function g_t that convert latent representation $\boldsymbol{o}_{\bar{t}_i}$ to a time budget $\bar{t}_i$. It uses a series of neural network layers to mimic the sampling operation from a deep continuous-time distribution. Specifically, $\boldsymbol{o}_{\bar{t}_i}$ is firstly transformed by a deconvolutional layer "DeConv()", then follow up with

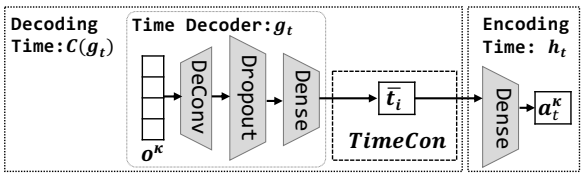


Figure 4: End-to-end time distributions inference by the proposed random time deep sampler, temporally-valid activation, and encoding of continuous times

a dropout layer "DropOut()" and another multi-perceptron layer "Dense()". Finally, it outputs a intermediate value of $\bar{t}_i$ before the time constraint functions C which is covered in the next Section 3.6. The mechanism is that the dropout layer can supply random noise. However, different from the conventional dropout layer which is only used during the training phase, our proposed dropout layer is used in both the training and the inference phases. The deconvolutional layer can increase stochasticity in this process. This series of neural network operation are summarized in Equation 2.

After the time constraint functions C, a realistic temporally-valid time value is generated and ready to be encoded to $\boldsymbol{a}_{\bar{t}_i}$ as LSTM inputs. In general, we use an encoding operation (encoding time function h_t in Figure 4) which is another dense layer that maps the generated time $\bar{t}_i$ back to the hidden vectors using $\boldsymbol{a}_{\bar{t}_i} = \text{Dense}(\bar{t}_i)$. It is also summarized in Equation 2 as follows:

$$\begin{aligned} \bar{t}_i &= C(\text{Dense}(\text{DropOut}(\text{DeConv}(\boldsymbol{o})))) \\ \boldsymbol{a}_{\bar{t}_i} &= \text{Dense}(\bar{t}_i) \end{aligned} \tag{2}$$

where "$C(\cdot)$" (corresponding to "TimeCon" in Figure 4) represents different types of time constraint functions which are further detailed in the next Section 3.6.

3.6 Temporally-valid activations for ensuring time constraints

Unlike conventional random walks, truncated temporal walks require that the temporal edges also satisfy various temporal-validity constraints encoded in $\mathbb{C}$ of Equation (1) which can be instantiated by the specific form of the term in Equation (2). Such constraints are implemented as auxiliary regularizing functions "TimeCon()" illustrated in Figure 4. Several methods are proposed to address various time-valid constraints in need, including clipping method, nested ReLU Bounding method, and Mini-max Bounding. Some of the constraints enforce the nonnegativeness, upper/lower bounds, and joint boundary of the time information of the decoded temporal edges. Specifically, 1) *Clipping*, which clips the generated time to boundary values if it is outside the range, is similar to a method commonly used in image generation [9]; 2) *Nested Relu Bounding* can ensure $0 \leq \bar{t}_i \leq \bar{t}_{i-1}$, which uses two nested *Relu* functions $\bar{t}_i = Relu(\bar{t}_i) - Relu(\bar{t}_i - \bar{t}_{i-1})$; and 3) *Mini-max Bounding* $\bar{t}_i = \text{MinMax}(t_i)$ uses a scaling based on "min" and "max" values within a sampled set of truncated sequences $\{\bar{s}\}$ for each training epoch. Initially, a "min-value-transform" operation is performed. A minimum value is obtained from this mini-batch: $min(\{\bar{t}_i\})$. Then, if $min(\{\bar{t}_i\}) \leq \epsilon$, then $\bar{t}_i = \bar{t}_i - min(\{\bar{t}_i\})$, otherwise, go to the

next step. Here, ϵ is a hyper-parameter with a small value (e.g. $\epsilon < 1e-3$) to prevent a zero value for $\bar{t}_i$. Next, the maximum value of the min-value-transformed mini-batch $max(\{\bar{t}_i\})$ is obtained. If $max(\{\bar{t}_i\}) > 1$, then, $t_i = \bar{t}_i / max(\{\bar{t}_i\})$, otherwise, nothing is done.

3.7 Discriminator LSTM-based classifier design

Discriminator $\mathcal{D}$ tries to distinguish real or generated walks. It utilizes a real temporal walk sampler (detailed in Appendix A.3) to extract real truncated temporal walks s, and uses generated truncated temporal walks from the generator $\mathcal{G}$. Then, the classifier of discriminator $\mathcal{D}$ is based on a recurrent architecture where recurrent units also adopt LSTM units. Each input is thus a truncated temporal walk $\bar{s}$ that consists of $x, \bar{t}_0, \bar{e}_1, \bar{e}_2, \cdots, y$ sequentially. We can directly leverage the same encoding operations introduced in Section 3.3 to encode them here and input encoded features into LSTM unit.

Training stopping criteria: TG-GAN uses an early-stopping mechanism that relies on one specific chosen evaluation metric, Mean degree in MMD distance (detailed in Section 4) to save time for the case of large graphs. The competitor methods use their respective default training mechanism.

Complexity analysis: TG-GAN is $O(L_s)$ because of its *time-budgeted temporal walks* assembly using a truncated temporal walk generator, where L_s is the maximal length of all the *time-budgeted temporal walks*. Memory complexity is $O(|V| \cdot L)$ for storing logit vectors of sampled nodes, where L is the length of *truncated temporal walks*. This makes our proposed model highly efficient for handling large graphs without any information loss. Here, our method developed in continuous times has a considerable advantage over other potential strategies using snapshots. All those require at least $O(|V|^2 \cdot T)$ and still suffer from information loss, where T is the number of time snapshots.

4 EXPERIMENTS

Synthetic datasets: datasets with increasing complexity were taken from scale-free random graphs [1], which are static graphs with a power-law degree distribution. They are generated by progressively adding nodes to an existing network and introducing links to existing nodes with preferential attachment, such that the probability of linking to a given node v is proportional to the number of existing links that node already has. We modified this generation by adding a time-dependent link and node generation step (cf. Appendix A.4). This method was used to generate three synthetic datasets with $\{100, 500, 2500\}$ nodes, respectively. For each dataset, different iterations of the simulation, i.e., $\{200, 100, 100\}$ graph samples, were used.

Real-world datasets: *a) User authentication graph.* Consists of the authentication activities of 97 users working on 27 computers or servers (nodes in the graph) in an enterprise intranet over a $485h$ period [18]. For this evaluation exercise, we focused on single user data. Each hour was treated as a temporal graph sample, and all timestamps were normalized to a range of $[0, 1]$. *b) Public transport graph.* Farecard records from the Washington D.C. metro system (91 stations as graph nodes) comprise millions of trip records with anonymized user information that takes the form <user id, entry station, timestamp, exit station, timestamp>. The dataset captures all trips taken over a period of three months (123 days) from May 2016 to July 2016, with each day being treated as a temporal graph sample. Since metro operations stop at 1am, we shifted all the timestamps by one hour to adhere to a $24h$ interval. All timestamps were converted to a $[0, 1]$ interval. All the datasets used were split into a separate 80% training and 20% test datasets. For the prescribed model, which required considerable main memory, sparser graph samples were used to adhere to memory limits.

Metrics: A simple means to evalute the various methods is to report the mean of specific graph measures based on a set of graph samples as shown in Table 1. A more rigorous evaluation method is Maximum Mean Discrepancy (MMD) [13], which evaluates the distribution distances of two sets of graph samples. One set is from a generative model and the other is real-world data. MMD is becoming more popular (cf. [6, 32]) than Kullback–Leibler divergence, or other distance metrics for two-sample tests of high-dimensional distributions [20]. Since MMD measures whole distributions, it handles uniqueness and diversity issues of high-dimensional samples well. MMD had to be used with different graph measurements on each graph sample. To ensure a fair comparison, both, continuous-time and discrete-snapshot graph measures [26] were used (check column names in Tables 2 and 3. Details are given in Appendix A.6).

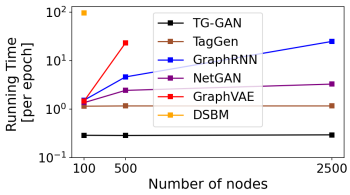
Competing methods: Our experimentation utilized five comparison methods: 1) TagGen [33], 2) GraphRNN [32], 3) NetGAN [4], 4) GraphVAE [25], and 5) a prescribed method, Dynamic-Stochastic-Blocks-Model (DSBM) [29]. We created snapshot graphs for these methods if needed, trained the models, and recovered the continuous time from the generated snapshot graphs (as detailed in Appendix A.5). Necessary parameter-tuning was done to ensure that TG-GAN performed as expected (cf. Appendix A.7).

4.1 Quantitative performance

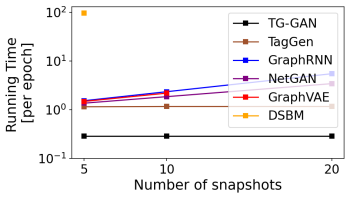
This section discusses the performance of TG-GAN in relation to the comparison methods. Given their scalability limitations, GraphVAE and DSBM were omitted for the 500-node and 2500-node datasets given their estimated runtime of several years for those cases.

Performance on real-world datasets through basic sample mean metrics. Table 1 presents the mean values of generated graph samples vs. real-world data. The performance of a model is assessed by how close its mean values for a generated set of graph samples are to the real-world data. For Authentication data, only TagGen outperforms our TG-GAN model in the Mean Degree metric. TG-GAN shows the best performance for all other metrics. For Metro data, TG-GAN outperforms all methods under all metrics. DSBM performs poorly in all cases. A more rigorous evaluation using MMD distance is presented in the next section.

Performance on synthetic datasets through MMD: Tables 2 and 3 present the MMD distances for the different graph measures for the three synthetic datasets (indicated as 100, 500, 2500 nodes in the first column). The lower the values, the better the respective performance for a method and dataset. Our TG-GAN method consistently outperforms all other methods. For the continuous-time graph measure (Table 2), including Average Degree, Mean Average Degree, and Mean Coordination Number, TG-GAN achieved a performance that was *two orders of magnitude better* than most of the comparison methods expect GraphRNN for the case of 500-node



(a) Running time V.S. Number of nodes



(b) Running time V.S. Number of snapshots

Figure 5: Running time experiments

Table 1: Continuous-time graph measures (the closer a value is to the real data's values, the better that model performed)

data	Metrics / Method	Mean Degree	Average Group Size	Average Group Number	Mean Coordination Number
Auth.	TagGen	**0.0166**	1.1624	23.2808	0.8726
	GraphRNN	0.0166	1.1154	24.2373	0.5856
	NetGAN	0.1116	2.2971	11.8050	11.1950
	GraphVAE	0.0204	1.1156	24.2360	0.5938
	DSBM	0.1918	0.9999	27.0000	2.2204
	TG-GAN	2.89e-05	**1.0184**	**26.5156**	**0.0359**
	Real	0.0166	1.0276	26.2911	0.0959
Metro	TagGen	0.0082	1.0285	88.4783	0.1089
	GraphRNN	0.0026	1.0109	90.0176	0.0239
	NetGAN	0.0068	1.0304	88.3149	0.1105
	GraphVAE	0.0026	1.0109	90.0160	0.0240
	DSBM	0.2544	0.9999	91.0000	2.22e-16
	TG-GAN	**0.00077**	**1.0072**	**90.3523**	**0.01425**
	Real	0.00065	1.0077	90.3012	0.0154

Table 2: Distances between real and generated graph sample sets using various continuous-time gragh measures in MMD (the lower the better)

data	Metrics / Method	Average Degree	Mean Average Degree	Group Size	Average Group Size	Mean Coordination Number	Mean Group Number	Mean Group Duration
Auth.	TagGen	0.0005	0.0015	0.9373	0.0184	0.3500	1.1737	0.0727
	GraphRNN	1.68e-05	1.42e-05	0.8053	0.0077	0.1829	0.9114	0.0654
	NetGAN	0.0036	0.0090	1.1192	1.0787	1.5065	1.4122	**0.04337**
	GraphVAE	1.70e-05	1.41e-05	0.8030	0.0077	0.1819	0.9083	0.0658
	DSBM	0.0002	0.0304	0.6344	0.0007	0.0087	0.2782	0.9315
	TG-GAN	**3.38e-09**	**2.94e-09**	**0.1187**	**0.0004**	**0.0047**	**0.1035**	0.1974
Metro	TagGen	3.78e-05	5.75e-05	1.4048	0.0004	0.0087	1.4556	0.5761
	GraphRNN	5.92e-06	3.82e-06	0.1826	1.00e-05	7.31e-05	0.0745	1.0376
	NetGAN	2.97e-05	3.85e-05	1.7542	0.0005	0.0091	1.6471	0.6608
	GraphVAE	5.88e-06	3.84e-06	0.1831	1.02e-05	7.54e-05	0.0754	1.0320
	DSBM	0.0004	0.0634	1.2656	5.99e-05	0.0002	0.4198	0.8011
	TG-GAN	**2.86e-08**	**1.45e-08**	**0.0065**	**2.92e-07**	**1.10e-06**	**0.0020**	**0.0910**
100	TagGen	5.73e-06	2.93e-05	1.3295	0.0142	1.2362	1.3434	0.0103
	GraphRNN	4.80e-05	8.16e-06	1.4152	0.0012	0.0019	1.4899	0.0342
	NetGAN	8.67e-05	0.0004	1.3785	0.0646	1.6637	1.5070	**0.0019**
	GraphVAE	7.72e-05	6.10e-06	1.4155	0.0012	0.0022	1.4933	0.0317
	DSBM	0.0083	0.0488	**0.9932**	1.8548	1.1320	**1.0690**	0.0020
	TG-GAN	**6.46e-07**	**1.35e-06**	1.24	**0.0004**	**0.0005**	1.3419	0.0041
500	TagGen	7.14e-05	0.0005	0.8500	0.0023	0.7982	0.8656	0.0079
	GraphRNN	7.42e-06	3.03e-06	1.3557	**0.0001**	**1.17e-05**	1.4287	0.0615
	NetGAN	2.14e-05	8.94e-05	0.8554	0.0162	1.4453	0.9566	**0.0001**
	GraphVAE	6.96e-06	2.46e-06	1.2291	0.0002	0.0002	1.2982	0.0227
	TG-GAN	**1.10e-06**	**2.10e-06**	**0.8000**	0.2312	1.8727	**0.0800**	0.2032
2500	TagGen	4.87e-06	3.66e-05	0.7156	0.0005	0.1413	**0.7382**	0.0078
	GraphRNN	1.80e-06	9.33e-07	1.0961	**0.0002**	**0.0002**	1.1154	0.78380
	NetGAN	1.48e-06	6.37e-06	**0.6464**	0.0017	1.3449	0.7414	**0.0003**
	TG-GAN	**4.78e-07**	**2.57e-07**	1.1189	**0.0002**	**0.0002**	1.1292	0.0377

data. This was largely due to the amount of information loss all these models suffered in order to perform a discrete-time snapshot. The most recent TagGen method achieved good performance in some of the metrics, such as Average Degree, Mean Average Degree. TG-GAN, GraphRNN and GraphVAE generally achieved a lower value than DSBM. This indicates that *deep generative methods perform better than existing prescribed models for the case of continuous-time measures.*

For discrete-time graph measures, the competitive advantage of TG-GAN is less clear. In some cases (Betweenness Centrality and Receive Centrality for 100-node graphs), DSBM achieved a better performance than deep generative methods. For 500-node and 2500-node data, TagGen outperforms all other methods. TG-GAN did, however, outperform the other two deep generative methods for the 100-node graphs, but this advantage was less significant for the 500-node graph. Overall, TG-GAN performed best for several measures, including Betweenness Centrality, Broadcast Centrality, Receive Centrality, and Temporal Correlation. The narrower performance gap for the larger synthetic graphs could be the result of our early-stopping mechanism, which is included to reduce overfitting.

Performance on real-world datasets through MMD: Tables 2 and 3 demonstrate the effectiveness of the proposed TG-GAN framework for real-world datasets (identified as Auth. and Metro in the first column). The overall performance characteristics differe from the synthetic datasets. TG-GAN outperforms all other methods in terms of all metrics and datasets. The only exception is Betweenness Centrality, for which NetGAN shows the best performance. Looking at how effective TG-GAN is for different graph sparsities, we observe

Table 3: Distances between real and generated graph sample sets using various discrete-time gragh measures in MMD (the lower the better)

Nodes	Metrics / Method	Betweenness Centrality	Broadcast Centrality	Burstiness Centrality	Closeness Centrality	Nodes' Temporal Correlation	Receive Centrality	Temporal Correlation
Auth.	TagGen	0.0159	0.6645	0.0926	0.0409	0.4797	0.5439	9.95e-04
	GraphRNN	4.41e-06	0.3071	0.2090	0.0223	0.0057	0.3072	8.26e-06
	NetGAN	**4.40e-06**	0.5996	0.0302	0.8119	0.0056	0.5971	8.27e-06
	GraphVAE	4.41e-06	0.3999	0.1718	0.0390	0.0057	0.3973	8.26e-06
	DSBM	0.9943	0.3598	0.0326	0.1016	0.0594	0.3628	0.0013
	TG-GAN	5.05e-05	**0.1468**	**0.0020**	**6.97e-04**	**0.0036**	**0.1365**	**5.23e-06**
Metro	TagGen	0.2260	NaN [2]	0.0114	1.79e-05	0.2994	NaN [2]	1.46e-04
	GraphRNN	0.0815	0.7316	0.0031	2.54e-04	NaN [2]	0.7351	NaN [2]
	NetGAN	0.0829	0.5946	0.0244	0.0166	NaN [2]	0.5811	NaN [2]
	GraphVAE	0.0829	0.7509	0.0030	1.99e-04	NaN [2]	0.7374	NaN [2]
	DSBM	0.7880	1.1403	0.0228	0.0164	0.0223	1.0444	2.01e-04
	TG-GAN	**0.0120**	**0.0257**	**0.0026**	**8.36e-06**	**2.86e-05**	**0.0266**	**4.95e-09**
100	TagGen	0.0591	0.6918	0.0035	**9.70e-05**	0.8301	0.2858	0.0014
	GraphRNN	0.9567	**0.1658**	0.3790	5.89e-04	0.0011	0.3023	1.81e-06
	NetGAN	0.6497	0.7058	0.0092	0.2073	0.0014	0.2878	**7.31e-07**
	GraphVAE	0.9567	0.2167	0.4138	5.37e-04	0.0011	0.3539	1.81e-06
	DSBM	**0.0020**	0.4016	0.0526	0.01666	0.0183	**0.1317**	1.33e-04
	TG-GAN	0.5606	0.2100	**0.0026**	0.0015	**0.0010**	0.2181	1.10e-06
500	TagGen	**0.0179**	0.5209	**0.0021**	- [1]	0.9394	0.1171	1.97e-04
	GraphRNN	0.7912	0.1556	0.1621	- [1]	0.0049	0.4241	1.75e-07
	NetGAN	0.7928	0.3253	0.0415	- [1]	0.0049	**0.0948**	1.75e-07
	GraphVAE	0.7928	**0.0871**	0.1921	- [1]	0.0049	0.2858	1.75e-07
	TG-GAN	0.7231	0.2878	0.0842	- [1]	**0.0048**	0.2087	**1.74e-07**
2500	TagGen	**0.0047**	0.4102	0.0038	- [1]	0.9976	0.1302	1.5e-05
	GraphRNN	0.8802	1.0239	**9.65e-07**	- [1]	0.0044	1.2410	1.00e-08
	NetGAN	0.8801	**0.1200**	0.0169	- [1]	0.0044	0.0965	1.00e-08
	TG-GAN	0.8245	0.1549	0.4296	- [1]	**0.0043**	**0.1979**	**9.76e-09**

[1] programs could not finished in three days.
[2] programs return errors.

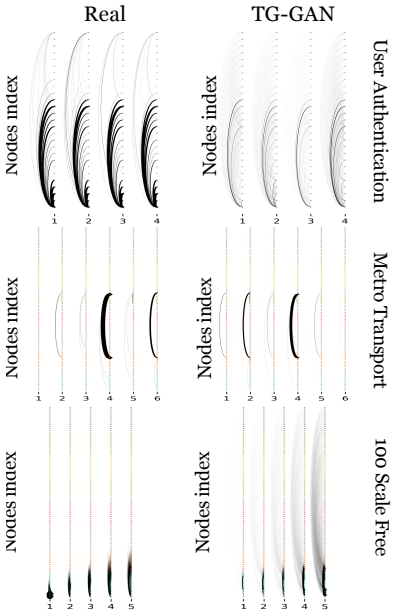


Figure 6: Comparisons of real graphs (left column) and graphs generated by TG-GAN (right column) through snapshot graphs.

that the authentication graph is sparser than synthetic scale-free graphs, while the metro graph is even sparser with only one or two edges per snapshot (typically 1-3 trips per day per traveler). We can see that for the discrete-time measures in Table 3, GraphRNN and GraphVAE have NaN values for Node Temporal Correlation, Temporal Correlation, or Receive Centrality for the sparse metro graph, since they fail to model empty graphs for most temporal snapshots (no trips happened during that snapshot). The same conclusion can also be drawn from the qualitative visualization in Appendix A.8.

In short, a highlight from the above results is the effectiveness of TG-GAN for different graph sparsities. The metro transport graph is an extremely sparse graph (typically 1-3 trips per day per traveler), while Authentication data has denser graphs with dozens of edges linked to one node. The synthetic scale-free graphs grow their density over time since more and more edges are added to such a graph. With increasing density, the advantage of TG-GAN decreases and we might need more training epochs so as to allow TG-GAN to better converge on an optimized state. Also, empty snapshots are challenging for many existing methods, but pose no challenge to TG-GAN. The following qualitative visualization provide further intuition on the graph generation properties of the different methods.

4.2 Qualitative analysis

The visualizations of Figure 6 show temporal patterns for the 1) user authentication graphs, 2) metro transport graphs, and 3) 100-node scale-free graphs. The y axis is the index of all nodes and the x axis is the index of the discrete-time snapshots. Arcs between two nodes represent a temporal edge. The darker an edge is, the more edges it represents based on all the graph samples (normalized values in a range from 0 to 1). Once again, TG-GAN performed is shown to perform well, since it captures the temporal patterns of the actual graphs the models are based on, i.e., similar arc patterns between the left and right column in Figure 6. Additional qualitative graph visualizations for all the other competing methods are given in Appendix A.8.

4.3 Running time analysis

The results of our running time experiments are shown in Figure 5. The running time is shown with respect to the growth of number of nodes numbers and time snapshots. All running times are in $log10$ scale for one epoch. Figure 5 (a) shows the running time as function of the number of nodes. Both, our TG-GAN method and TagGen have a constant running in terms of number of nodes, which is especially important when dealing with large graphs. GraphRNN and NetGAN with their quadratic-time complexity are still able to run for 2500-node graphs. Other methods, such as GraphVAE and DSBM would only run for up to 500 and 100 node graphs, respectively. We also experimented with different numbers of discrete time snapshots (cf. Figure 5 (b)). Both TG-GAN and TagGen again exhibit constant running time with the number of snapshots. GraphRAN and NetGAN again exhibit exponential growth. These results demonstrate the advantage of TG-GAN for large temporal graphs.

5 CONCLUSIONS

To effectively model the generative distributions of temporal graphs and retain continuous-time information, this work proposes the

first-of-its-kind Temporal Graph Generative Adversarial Network (TG-GAN) framework. Our new framework learns the representations of temporal graphs via truncated temporal walks. It includes a novel temporal generator to model truncated temporal walks with profile information that also takes a strong temporal diffusion process and temporal constraints into account. Extensive experiments with synthetic and real-world datasets demonstrate the advantages of TG-GAN over existing deep generative and prescribed models.

REFERENCES

[1] Albert-László Barabási, Réka Albert, and Hawoong Jeong. 1999. Mean-field theory for scale-free random networks. *Physica A: Statistical Mechanics and its Applications* 272, 1-2 (1999), 173–187.

[2] Edward A Bender and E Rodney Canfield. 1978. The asymptotic number of labeled graphs with given degree sequences. *Journal of Combinatorial Theory, Series A* 24, 3 (1978), 296–307.

[3] Z Bahrami Bidoni, R George, and KA Shujaee. 2014. A Generalization of the PageRank Algorithm. ICDS.

[4] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. 2018. Netgan: Generating graphs via random walks. *arXiv preprint arXiv:1803.00816* (2018).

[5] Béla Bollobás, Christian Borgs, Jennifer Chayes, and Oliver Riordan. 2003. Directed scale-free graphs. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 132–139.

[6] Wacha Bounliphone, Eugene Belilovsky, Matthew B Blaschko, Ioannis Antonoglou, and Arthur Gretton. 2015. A test of relative similarity for model selection in generative models. *arXiv preprint arXiv:1511.04581* (2015).

[7] Marco Corneli, Pierre Latouche, and Fabrice Rossi. 2016. Exact ICL maximization in a non-stationary temporal extension of the stochastic block model for dynamic networks. *Neurocomputing* 192 (2016), 81–91.

[8] P ERDdS and A R&wi. 1959. On random graphs i. *Publ. Math. Debrecen* 6, 290-297 (1959), 18.

[9] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. 2015. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576* (2015).

[10] Anna Goldenberg, Alice X Zheng, Stephen E Fienberg, Edoardo M Airoldi, et al. 2010. A survey of statistical network models. *Foundations and Trends® in Machine Learning* 2, 2 (2010), 129–233.

[11] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

[12] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. 2018. Dyngem: Deep embedding method for dynamic graphs. *arXiv preprint arXiv:1805.11273* (2018).

[13] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. 2012. A kernel two-sample test. *Journal of Machine Learning Research* 13, Mar (2012), 723–773.

[14] Xiaojie Guo and Liang Zhao. 2020. A Systematic Survey on Deep Generative Models for Graph Generation. arXiv:2007.06686 [cs.LG]

[15] Till Hoffmann, Mason A Porter, and Renaud Lambiotte. 2013. Random walks on stochastic temporal networks. In *Temporal Networks*. Springer, 295–313.

[16] Petter Holme. 2015. Modern temporal network theory: a colloquium. *The European Physical Journal B* 88, 9 (2015), 234.

[17] Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144* (2016).

[18] Alexander D. Kent. 2015. Cybersecurity Data Sources for Dynamic Network Research. In *Dynamic Networks in Cybersecurity*. Imperial College Press.

[19] Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013).

[20] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.

[21] Giang H Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. 2018. Dynamic network embeddings: From random walks to temporal random walks. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 1085–1092.

[22] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 701–710.

[23] Polina Rozenshtein and Aristides Gionis. 2019. Mining Temporal Networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 3225–3226.

[24] Hooman Peiro Sajjad, Andrew Docherty, and Yuriy Tyshetskiy. 2019. Efficient representation learning using random walks for dynamic graphs. *arXiv preprint arXiv:1901.01346* (2019).

[25] Martin Simonovsky and Nikos Komodakis. 2018. Graphvae: Towards generation of small graphs using variational autoencoders. In *International Conference on Artificial Neural Networks*. Springer, 412–422.

[26] Ann E Sizemore and Danielle S Bassett. 2018. Dynamic graph metrics: Tutorial, toolbox, and tale. *NeuroImage* 180 (2018), 417–427.

[27] Larry Wasserman. 2013. *All of statistics: a concise course in statistical inference*. Springer Science & Business Media.

[28] Kevin Xu. 2015. Stochastic block transition models for dynamic networks. In *Artificial Intelligence and Statistics*. 1079–1087.

[29] Kevin S Xu and Alfred O Hero. 2014. Dynamic stochastic blockmodels for time-evolving social networks. *IEEE Journal of Selected Topics in Signal Processing* 8, 4 (2014), 552–562.

[30] Min Yang, Junhao Liu, Lei Chen, Zhou Zhao, Xiaojun Chen, and Ying Shen. 2019. An Advanced Deep Generative Framework for Temporal Link Prediction in Dynamic Networks. *IEEE Transactions on Cybernetics* (2019).

[31] Tianbao Yang, Yun Chi, Shenghuo Zhu, Yihong Gong, and Rong Jin. 2011. Detecting communities and their evolutions in dynamic social networks—a Bayesian approach. *Machine learning* 82, 2 (2011), 157–189.

[32] Jiaxuan You, Rex Ying, Xiang Ren, William L Hamilton, and Jure Leskovec. 2018. Graphrnn: Generating realistic graphs with deep auto-regressive models. *arXiv preprint arXiv:1802.08773* (2018).

[33] 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 International Conference on Knowledge Discovery & Data Mining*. 401–411.

A APPENDIX

A.1 Decoding and encoding operations for categorical data in the generator

As shown in Figure 2 (a), sampling categorical data includes sampling nodes with $g_v(\boldsymbol{o}_v)$, sampling starting flags $g_x(\boldsymbol{o}_x)$, and sampling ending flags $g_y(\boldsymbol{o}_y)$. Each of these share the similar categorical distribution sampling methods, and categorical data encoding procedures h_v, h_x using embedding layers to get features $\boldsymbol{a}_x, \boldsymbol{a}_{v_i}$ as inputs for the next LSTM unit. All the operations are summarized in Equation 3 as follows:

$$
\begin{aligned}
&g_v(\cdot) : \boldsymbol{v}_i \sim Cat(\boldsymbol{q}_{v_i}), \quad \boldsymbol{q}_{v_i} = \boldsymbol{W}_{v,up}\boldsymbol{o}_{v_i} + \boldsymbol{b}_{v,up} \\
&h_v(\cdot) : \boldsymbol{a}_{v_i} = \mathrm{Dense}(\boldsymbol{W}_{v,down}\boldsymbol{v}_i) \\
&g_x(\cdot) : \boldsymbol{x} \sim Bern(\boldsymbol{q}_x), \quad \boldsymbol{q}_x = \boldsymbol{W}_{x,up}\boldsymbol{o}_x + \boldsymbol{b}_{x,up} \\
&h_x(\cdot) : \boldsymbol{a}_x = \mathrm{Dense}(\boldsymbol{W}_{x,down}\boldsymbol{x}) \\
&g_y(\cdot) : \boldsymbol{y} \sim Bern(\boldsymbol{q}_y), \quad \boldsymbol{q}_y = \boldsymbol{W}_{y,up}\boldsymbol{o}_y + \boldsymbol{b}_{y,up}
\end{aligned}
\tag{3}
$$

where $\boldsymbol{q}_{v_i} \in \mathbb{R}^{|V|}, \boldsymbol{q}_x \in \mathbb{R}^2, \boldsymbol{q}_y \in \mathbb{R}^2$ are logits of categorical or Bernoulli distribution [27] for sampling a node, and $\boldsymbol{o}_{v_i} \in \mathbb{R}^H, \forall H \ll |V|$ is the output of the corresponding LSTM units, which is a highly concise representation that largely reduces the computing overhead especially for large graphs. $\boldsymbol{W}_{v,down} \in \mathbb{R}^{|V|\times H}, \boldsymbol{W}_{x,down} \in \mathbb{R}^{2\times 1}$ are different embedding matrices. Notice that we do not need to embed end flag y.

Here are some more details. Taking the operations for a node as an example, the second LSTM unit produces an output vector $\boldsymbol{o}_{v_i} \in \mathbb{R}^H$. Here, $H \ll |V|$ is the dimension of the embedding space of a node from V. So $\boldsymbol{o}_{v_i}$ is a highly concise representation that largely reduces the computing overhead especially for large graphs. Then $\boldsymbol{o}_{v_i}$ is projected upscale to another vector $\boldsymbol{q}_{v_i} \in \mathbb{R}^{|V|}$, which is a logit parameter of a categorical distribution for sampling a node $v_i \sim Cat(\boldsymbol{q}_{v_i})$. The projection function is defined as an affine transformation $\boldsymbol{W}_{up}\boldsymbol{o}_\kappa + \boldsymbol{b}_{up}$. The procedures for starting and ending flags are the same as the above except that the dimension of the decoded value is two, namely $\boldsymbol{q}_x, \boldsymbol{q}_y \in \mathbb{R}^2$, and hence the sampling of the starting and ending flags can be denoted as $x \sim Bern(\boldsymbol{q}_x)$ or

$y \sim Bern(\boldsymbol{q}_y)$, where *Bern* denotes the Bernoulli distribution [27]. $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{v}_i$ are one-hot vectors in this context.

Figure 2 (a) shows the encoding operations for categorical data. Besides the memory states, the decoded categorical data (i.e., either node v_i or starting flag x) in the last time step is also encoded for the next LSTM unit's inputs. Specifically, to convert categorical data including x, v_i, embedding layers are used in the form of two embedding matrices $\boldsymbol{W}_{x,down} \in \mathbb{R}^{2\times H}, \boldsymbol{W}_{v,down} \in \mathbb{R}^{|V|\times H}, \forall H \ll |V|$. H are the dimensions of the embedded vectors. All nodes share the same embedding parameters. Embedded vectors are passed to two different dense layers and generate input vectors $\boldsymbol{a}_x, \boldsymbol{a}_{v_i}$ for the next LSTM unit.

A.2 Gumbel-Max tricks for re-parameterization of categorical data

The sampling operation $\sim$ from categorical (or Bernolli) distributions in Equation 3 poses significant challenges to backpropagation training, which inherently requires differentiable objective functions to work. To address this issue, we leverage a recent re-parametrization trick based on Gumbel-Max [17]. More details can be found in [17]. Gumbel-Max re-parameterization of categorical distributions [17] creates $v_i' = \tanh((\boldsymbol{q}_{v_i} + \boldsymbol{g})/\tau)$, where τ is the so-called "temperature" hyper-parameter. Each value g_i in $\boldsymbol{g}$ is an independent and identically distributed (i.i.d.) sample from standard Gumbel distribution [17]. We can now generate a one-hot vector representation $\boldsymbol{v}_i$, whose i'th element is one and all the others are zeros, where $i = \arg\max_i \boldsymbol{v}_i'$. In this way, gradients can be back-propagated through $\boldsymbol{v}_i'$. The same approach is also used for start and end flags x, y. Notice that when τ becomes larger, the sampled values become more uniformly regulated, with a more stable gradient flow. The typical approach is to decrease τ as training continues and we can thus adopt a decrease strategy similar to [17] (Equations 4).

$$
\begin{aligned}
&\text{for nodes } v_i, && \boldsymbol{q}_{v_i} = \boldsymbol{W}_{v,up}\boldsymbol{o}_{v_i} + \boldsymbol{b}_{v,up} \\
&&& \boldsymbol{v}_i' = \tanh\left((\boldsymbol{q}_{v_i} + \boldsymbol{g}_{v_i})/\tau\right), v_i = \text{OneHot}(\arg\max \boldsymbol{v}_i') \\
&\text{for start indicator } x, && \boldsymbol{q}_x = \boldsymbol{W}_{x,up}\boldsymbol{o}_x + \boldsymbol{b}_{x,up} \\
&&& \boldsymbol{x}' = \tanh\left((\boldsymbol{q}_x + \boldsymbol{g}_x)/\tau\right), x = \text{OneHot}(\arg\max \boldsymbol{x}') \\
&\text{for end indicator } y, && \boldsymbol{q}_y = \boldsymbol{W}_{y,up}\boldsymbol{o}_y + \boldsymbol{b}_{y,up} \\
&&& \boldsymbol{y}' = \tanh\left((\boldsymbol{q}_y + \boldsymbol{g}_y)/\tau\right), y = \text{OneHot}(\arg\max \boldsymbol{y}')
\end{aligned}
\tag{4}
$$

A.3 Truncated temporal walk sampler

In the case of a temporal graph, a time-variant graph topology changes dynamically and a variable-length walk sequence can be sampled. However, if we truncate this variable-length sequences to small fixed-length sequences, there are two challenges: 1) subsequences of a temporal sequence might contain less information and are more challenging for finding patterns, older parts are more difficult to learn; and 2) sparse connections of different sub-graphs in a whole graph prevent the learning of global information. In the extreme cases, walkers could be stuck in a sub-graph with no out-links to the whole graph. This is referred to as the "Spider-Trap" problem in graph data mining [3].

To address the above challenges, we propose a novel sampling method for *truncated temporal walks* that first determines the starting edge of the walk and then samples the next edges sequentially following a temporal process. The proposed truncated temporal walk sampler is shown in Algorithm 1. Line 1 normalize times so that $t_{end} = 1$. Line 2 specifies the selection of a graph sample, while Lines 3-10 sample the profiles and walk sequences. The details of the starting edge sampler $\mathcal{K}$ and next edge sampler $\mathcal{H}$ are as follows:

Algorithm 1: Truncated temporal walks sampler

Data: $E = \{E_d\}, \forall E_d = \{\bar{e}_i(u_i, v_i, \bar{t}_i)\}, t_{end}, L$
Result: a set of truncated sequences: $\{(x, t_0, \bar{e}_1, \ldots, \bar{e}_L, y)\}$
1 initialize $\bar{t}_i \leftarrow \bar{t}_i / t_{end}$
2 sample $d \sim Uniform(1/|E|)$, and get E_d
3 $\bar{e}_{i_0}(v_{i_0}, u_{i_0}, \bar{t}_{i_0}) \sim \mathcal{K}(E_d)$
4 **if** $i == 1$ **then** $x \leftarrow 1, \bar{t}_0 \leftarrow 1$ **else** $x \leftarrow 0, \bar{t}_0 \leftarrow \bar{t}_{i-1}$
5 $i \leftarrow i_0$
6 **while** $i \le i_0 + L$ **do**
7 $\quad \bar{e}_{i+1}(u_{i+1}, v_{i+1}, \bar{t}_{i+1}) \sim \mathcal{H}(\bar{e}_i)$
8 $\quad i \leftarrow i + 1$
9 **end**
10 **if** $i == |E_d|$ **then** $y \leftarrow 1$ **else** $y \leftarrow 0$

Starting edge sampler $\mathcal{K}$: Several alternatives could be used here, such as a uniform distribution: $p_{\mathcal{K}}(e) = 1/|E|$. However, it is reasonable to use a distribution that is biased towards the start time, so starting edges that happen earlier have a higher probability. Biased ones are better candidates for extremely sparse graphs with very short temporal sequences. To build a biasd sampler, we could leverage a linearly-biased distribution: $p_{\mathcal{K}}(e) = t_i / \sum_{e_i \in E} t_i$ or exponential distribution: $p_{\mathcal{K}}(e) = exp(t_i)/\sum_{e_i \in E} exp(t_i)$.

Descendant edge sampler $\mathcal{H}$ with temporal jumps: A descendant edge following the current edge could be selected from among its adjacent edges either uniformly or considering the time decay. By extending the notion of jumps, we propose the use of temporal jumps, acted like a prior distribution, which should help achieve "smoothness" in posterior distributions. For example, in human mobility in metro networks can be modeled as a temporal graph in which a traveler could do walks between different metro stations along the transport network (i.e., temporal edges). Temporal jumps occur when a traveler switches between different modes of transport, e.g., walking, taxiing, etc. To provide for more robust and flexible modeling, we propose the use of "teleported temporal edges" with monotonically-increasing time stamps, by incorporating a Bayesian prior into the temporal graphs. Teleported temporal edges are a Bayesian prior-enhanced categorical distribution $Cat(p_{\mathcal{H}})$ with a probability of selecting the next edge that depends on time, and are implemented with an exponential function that decays with time and has a uniform distribution over all the nodes except the current node.

$$
e_i \sim \mathcal{H}(e_i|\boldsymbol{m}), \;\; m_i = \alpha\left(exp(t_i)/\sum_{j=i}^{|E|} exp(t_j) + 2\epsilon/(|V|-1)\right)
$$

where α is a normalization term to ensure all probabilities sum to 1, and ϵ is a very small teleport probability over all the other nodes except the current node.

A.4 Temporal scale-free random graph simulation

A directed scale-free graph [5] create a new edge from a new in-node, an existing node, or a new out-node by sampling a multi-normial distribution of three probabilites $\langle \alpha, \beta, \gamma \rangle, \forall \alpha + \beta + \gamma = 1$, which is adopted in Networkx [1] library. We modify this edge generation procedure to a temporal dependent generation. The general idea is to append a continuous-time value to generated edge in each constructing step. First, a uniform distribution within $[0, 1]$ is used to sample a time value, $t \sim Uniform()$. And, we know that Unifrom distribution output value from 0 to 1. By comparing t with cumulative probability $\langle \alpha, \alpha + \beta, 1 \rangle$, if $t \in [0, \alpha]$ a new in-node indexed as $|V| + 1$ is added and an exiting node is chosen from V with probability $p(v = v_i)$ as an out-node, where d_{in} is the function to get degree of v_i, δ_{in} is a hyper-parameter, r is a uniformly random generated real number. A temporal edge is created for them with time-stamp t. If $t \in (\alpha, \alpha + \beta]$, two existing nodes u_i and v_i are chosen, and a temporal edge is created with time t. If $t \in (\alpha + \beta, 1]$, a new out-node is got, and a temporal edge to a chosen exiting node is created with time t. The choice of existing in-node $p(u = u_i) = \frac{d_{in}(u_i) + \delta_{in}}{|E| + \delta_{in} r}$, where d_{in} is the function to get normalized in-degree of u_i, δ_{in} is a hyper-parameter, r is a uniformly-random-generated real number. The choice of existing out-node $p(v = v_i) = \frac{d_{out}(v_i) + \delta_{out}}{|E| + \delta_{out} r}$, where d_{out} is the function to get normalized out-degree of v_i, δ_{out} is a hyper-parameter, r is another uniformly-random-generated real number. For more details, check [5]. After each constructing step, the ellapsed time is cumulated. This process is terminated untill either number of edges is equal to number of nodes, or a preseted maximum time range is reached. And, we borrow part of the codes from Networkx [2] library's original Scale-free graph codes.

A.5 Competing methods details

Modifications of adapting competing methods developed for static graphs to temporal graphs are described as follows:

TagGen. This is a most recent work of temporal graph generation with deep generative method. However, it is developed for a specific type of temporal graph, temporal interaction network. Also, it can only handle fixed amount of timestamps which already exist in real data. Its design is scalable to very large graphs. We use the default parameters provided in the TagGen code.

GraphRNN. This is a recent state-of-the-art deep generative method. It is developed for a set of static graph samples. And, it is scalable to very large graphs. We use the default parameters provided in the GraphRNN code.

NetGAN. This is a recent deep generative method via random walks. It is developed for one static graph. And, it is scalable to very large graphs. We use the default parameters provided in the NetGAN code, but we change the random walks sampling code to be randomly sampling one graph from a set of graphs first before sampling random wals from graphs.

GraphVAE. This is also a secent development deep generative method. It is also targeted for static graphs. Its complaxity analysis makes it only runnable for small graphs. The original paper do not have a published code, so the code in GraphRNN paper is used. Also, default parameters are used.

DSBM. It is most recent development of prescribed models for dynamic graphs based on Stochastic Blocks Models. It utilizes a Markovian transition to model the dynamic in temporal graphs.

Our models can adapt to start-time and end-time easily by adding additional LSTM cell for each temporal edge, if other applications need end-time evaluation. The performance of our new TG-GAN framework was compared with the above models. Another question in those five models is how to convert the snapshots back to continuous time. We simply choose the middle point of time in each snapshot as the real time of a generated temporal edge. Also, notice that temporal edges are modeled as a time point for all the datasets and generated data. For continuous-time measures, the existing of temporal edges have a start time and end time. For simplicity, we assume a constant time for all temporal edges to exist. This assumption is also true for real-world authentication graph which only has one timestamp for each edge. For transport graph, the existing time of a temporal edge is the travel time from one station to another station, which is almost fixed in metro schedules and relatively small compared to a whole 24 hours. It is also a reasonable assumption to use the same small edge existing time.

A.6 MMD

In short, MMD measures how the distribution of one set of samples is similar to another set of samples. Given $\boldsymbol{X} \in \mathbb{R}^{n \times k}$, each row $\boldsymbol{X}_{i,}$ is a sampled vector from a unknown distribution. There is another $\boldsymbol{X'} \in \mathbb{R}^{n' \times k}$, each row $\boldsymbol{X'}_{i,}$ is a sampled vector from another unknown distribution. We have $MMD(\boldsymbol{X}, \boldsymbol{X'})$ as a distance measurement of these two sample sets. $MMD(\boldsymbol{X}, \boldsymbol{X'}) = 0$ means two sets are exactly the same. In this experiment, different empirical graph measures are chosen for $\boldsymbol{X}$. For example, the continuous-time average degree distribution of one graph sample is $\boldsymbol{X}_{i,} \in \mathbb{R}^{1 \times |V|}$, where $|V|$ is the number of nodes. We can compute $MMD(\boldsymbol{X}, \hat{\boldsymbol{X}})$ to see if graphs samples generated from a trained model $\boldsymbol{X}$ is close to real graph samples $\hat{\boldsymbol{X}}$.

A.7 Parameter-tuning

These includes a set of hyper-parameters that can be tuned for TG-GAN to achieve the best performance. Here are the list: learning rate [0.003, 0.003], generator node embedding size [node number / 2], discriminator node embedding size [node number / 2], $L2$ penalty of discriminator $[5e-5]$, $L2$ penalty of generator $[1e-7]$, up-project of x, y [16-64], up-project of t [32-128], up-project of node v [32-128], generator LSTM cell state c, h $[[100, 20], [50, 10], [100], [50]]$, discriminator LSTM cell state c, h $[[80, 20], [40, 10], [80], [40]]$, start temperature of gumbel-max [5], wasserstein penalty [1, 10], time decoding methods [Gaussian, Gamma, Beta, Deep random time sampler], time constraint activation methods [Minmax, clipping, Relu], initial noise type [Uniform, Gaussian]. More potential hyper-parameter are released in github in the future.

A.8 Additional experimental results

Figure 7 shows temporal patterns for the various generated graphs and compares them to the actual graph. The y axis is the index of all nodes and the x axis is the index of the discrete-time snapshots.

[1] https://networkx.github.io/documentation/stable/index.html
[2] https://networkx.github.io/documentation/stable/index.html

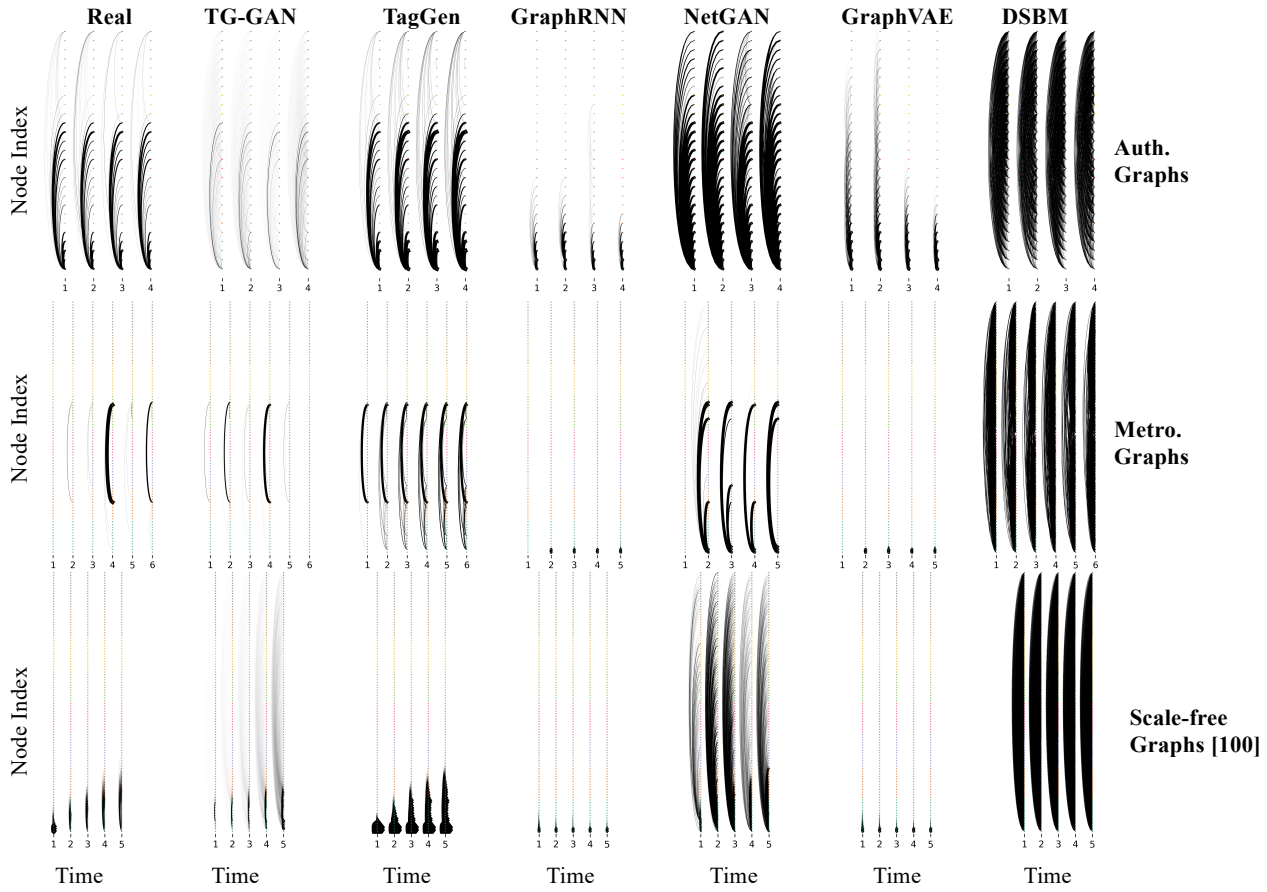


Figure 7: Comparison of the real graphs (left), TG-GAN and five comparison methods for three different datasets

Arcs between two nodes represent a temporal edge. A comparison across graphs is facilitated b the small dots on each column representing nodes as well. The dots repeat across all temporal snapshots. The figures are created by, first, converting each graph sample to snapshots, and then, for each snapshot, summing up counts of an edge for all graph samples. The darker an edge is, the more edges it represents based on all the graph samples (normalized values in a range from 0 to 1).

For user authentication graphs, we observe that this graph is densely connected in sub-regions of the whole graph. TG-GAN mimics the real graph topology quite well. GraphRNN and GraphVAE show a trend towards similar topology. More training could potentiall result in better performance. DSBM does not create a graph close to the actual topology.

For the metro transport graph,a good example of an extremely sparse graph, typically one edge has only one snapshot. Most of the temporal edges happen in one or two snapshots. We can see that TG-GAN is the only model to capture this challenging sparse graphs.

For 100-node scale-free synthetic graph, we can see that edges are connected increasingly as time grows. First, it did agree with typical scale-free graph pattern though it is a time-dependent growth. Then, TG-GAN performs the best to mimic this growing connection pattern. GraphRNN and GraphVAE also capture the overall trends. Probably, more training would improve their performance. DSBM do not perform well too.