DynG2G: An Efficient Stochastic Graph Embedding Method for Temporal Graphs

Mengjia Xu, Apoorva Vikram Singh, and George Em Karniadakis

Abstract—Dynamic graph embedding has gained great attention recently due to its capability of learning low-dimensional and meaningful graph representations for complex temporal graphs with high accuracy. However, recent advances mostly focus on learning node embeddings as deterministic "vectors" for static graphs, hence disregarding the key graph temporal dynamics and the evolving uncertainties associated with node embedding in the latent space. In this work, we propose an efficient stochastic dynamic graph embedding method (DynG2G) that applies an inductive feedforward encoder trained with node triplet energy-based ranking loss. Every node per timestamp is encoded as a time-dependent probabilistic multivariate Gaussian distribution in the latent space, and, hence, we are able to quantify the node embedding uncertainty on-the-fly. We have considered eight different benchmarks that represent diversity in size (from 96 nodes to 87 626 and from 13 398 edges to 4 870 863) as well as diversity in dynamics, from slowly changing temporal evolution to rapidly varying multirate dynamics. We demonstrate through extensive experiments based on these eight dynamic graph benchmarks that DynG2G achieves new state-of-the-art performance in capturing the underlying temporal node embeddings. We also demonstrate that DynG2G can simultaneously predict the evolving node embedding uncertainty, which plays a crucial role in quantifying the intrinsic dimensionality of the dynamical system over time. In particular, we obtain a "universal" relation of the optimal embedding dimension, L_o, versus the effective dimensionality of uncertainty, D_u, and infer that $L_o = D_u$ for all cases. This, in turn, implies that the uncertainty quantification approach we employ in the DynG2G algorithm correctly captures the intrinsic dimensionality of the dynamics of such evolving graphs despite the diverse nature and composition of the graphs at each timestamp. In addition, this $L_0 - D_u$ correlation provides a clear path to selecting adaptively the optimum embedding size at each timestamp by setting $L \geq D_u$.

Index Terms—Dynamic graph, graph embedding, multivariate Gaussian distribution, uncertainty quantification.

Mengjia Xu is with the Division of Applied Mathematics, Brown University, Providence, RI 02912 USA, and also with the McGovern Institute for Brain Research, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: mengjia@mit.edu).

Apoorva Vikram Singh is with the Department of Electrical Engineering, National Institute of Technology, Silchar, Assam 788010, India (e-mail: singhapoorva388@gmail.com).

George Em Karniadakis is with the Division of Applied Mathematics, Brown University, Providence, RI 02912 USA (e-mail: george_karniadakis@brown.edu).

I. Introduction

NUMEROUS real-world systems encompassing complex behavior and interaction among entities can be naturally modeled as "graphs" [1]. Recently, deep neural network-based graph embedding techniques have proven very promising and effective in learning highly informative graph representations for various graph analytics problems in diverse fields [2], e.g., social network analysis [3], protein–protein interaction prediction [4], functional brain network analysis [5], [6], electronic health record data analysis [7], molecular property prediction in chemistry [8], tumor microenvironment staging for prognosis prediction of the gastric cancer [9], and high-energy physics data analysis [10]. Specifically, by leveraging neural networks, such as SkipGram [11], autoencoder [12], and graph convolutional networks (GCNs) [13], high-dimensional graphs can be effectively transformed into low-dimensional latent space as continuous vectors (i.e., node embeddings), while the original graph topological properties are maximally preserved. Moreover, the node embedding distances in the latent space can be utilized to approximate the node similarity in the original irregular space. To this end, the obtained graph embeddings can benefit a variety of downstream graph analysis tasks, e.g., node classification, link prediction, anomaly detection, and recommender systems. Nevertheless, many of the aforementioned high-dimensional graphs in different applications exhibit heterogeneous evolving topologies and varying features at the temporal scale. Hence, temporal graph embedding methods have surged to become a hot topic due to their key advantages in capturing graph representations at both spatial and temporal scales. However, how to efficiently embed the dynamic graphs still remains quite challenging requiring to address the following three main questions.

1) How to stabilize the graph embedding training caused by heterogeneous graph dynamics in both topology and feature occurred in the temporal graphs?
2) How to estimate the uncertainty for node embeddings so that it provides robustness for fine-grained quantification of node properties in the latent space?
3) How to reduce computational cost and enhance scalability for graphs.

To tackle the aforementioned issues, we present a new stochastic graph embedding method (DynG2G) for learning probabilistic temporal graph representations and enabling node uncertainty quantification. The proposed DynG2G model follows the four main steps: 1) sampling node triplets based on the k-hop neighborhood incorporating edge weights;

2) developing an adaptive feedforward encoder to learn graph embeddings for temporal graphs with varying node numbers in different graph snapshots[1]; 3) initializing the deep encoder for the current timestamp with the learned hyperparameters from the previous graph snapshots; and 4) training and optimizing the neural network with a triplet energy-based square-exponential contrastive loss.

To demonstrate the effectiveness of DynG2G, we test it for temporal link prediction on eight different benchmarks, varying in size of nodes and edges but also varying in the intrinsic dynamics from very stable graphs to rapidly changing graphs as a function of time. Comparisons with baselines, when available, demonstrate that DynG2G can achieve favorable predictive performance and high efficiency in temporal link prediction on different benchmarks by using the obtained dynamic probabilistic node embeddings (i.e., multivariate Gaussian distribution in terms of mean and variance vectors). In addition, DynG2G provides an important capability of uncertainty quantification for the node embedding, which allows us to produce more reliable and quantitative graph representations in the latent space for various downstream tasks. Moreover, it reveals the (possibly evolving) effective dimensionality of the complex dynamical system that we represent as we demonstrate in our experiments for all eight benchmarks. This, in turn, relates to the optimum embedding size, which for problems with highly transient dynamics may be changing over time.

The main contributions of this article are given as follows.

1) We developed a stochastic graph embedding model (DynG2G) that projects temporal graphs from irregular domain to low-dimensional "function" space, which allows to capture node embedding uncertainty.
2) DynG2G produces more informative node embeddings that achieve superior performance in temporal link prediction task, for all benchmarks and especially for the highly dynamic benchmarks, e.g., UC Irvine Messages (UCI) and Bitcoin-OTC (Bit-OTC) datasets, see details in Section IV-B.
3) We discovered a "universal relation," a correlation that connects the optimum embedding dimension with the effective dimension of the dynamic system, hence identifying a clear way to select the optimum embedding dimension using uncertainty quantification.

We have posted our DynG2G code[2] on GitHub so that the interested reader can reproduce the results.

II. RELATED WORK

Dynamic graph embedding is a key and prerequisite step for temporal graph machine learning. It has attracted an increasing interest in dynamic graph embedding studies recently due to its superiority in learning latent graph representations for many different tasks. There are four main types of approaches.

---

[1]The number of the hidden units in our DynG2G encoder can be adaptively assigned based on the changing node numbers in different graph snapshots.

[2]https://github.com/GraceXu182/DynG2G

A. Matrix Factorization-Based Models

For this approach, Zhu *et al.* [14] proposed the DHPE model that mainly applies the eigendecomposition to construct the high-order proximity matrix of the network and then dynamically updates the node embedding of the next snapshot via the matrix perturbation approach [15]. However, matrix factorization-based models usually suffer from high computational complexity for large-scale temporal graphs; moreover, the incremental matrix decomposition procedure is prone to error accumulation. Zhang *et al.* [16] proposed another more advanced TIMERS model, which effectively reduces the error accumulation problem by using an error bound-based constraint during node embedding updating over time.

B. SkipGram-Based Models

The SkipGram model (also known as word2vec) was originally developed as a random walk-based language embedding approach, and it has been very successfully applied in diverse static graph embedding models, e.g., node2vec [17] and deepwalk [18]. The majority of dynamic graph embedding approaches focus on leveraging the conventional static graph embedding models incorporating temporal information [19]–[22]. In dynnode2vec, Mahdavi *et al.* [20] adopted the learned parameters in the previous snapshot for network initialization in the next snapshot. They trained the new evolving random walks and updated the node embedding for the evolving nodes at each time step. Hence, dynnode2vec can take advantage of the previously learned mappings to incorporate the important temporal dependencies features to predict the links in the next time step t; dynnode2vec [20] outperforms the conventional static graph embedding model (i.e., node2vec [17]) for anomaly detection. In addition to learning embeddings for discrete graph snapshots using conventional static graph embedding models, the CTDNE model [23] employs a continuous-time dynamic graph embedding approach, which enables learning of the time-preserving graph embeddings using the SkipGram model with valid temporal walks (W_T) dynamic graph streams (a scale of seconds or milliseconds).

C. Autoencoder-Based Models

The unsupervised learning-based deep autoencoder has shown great effectiveness in learning static graph embeddings and is particularly applicable to graph reconstruction problems. Goyal *et al.* [24] proposed a "semisupervised" dynamically expansion autoencoder model (DynGEM) for discrete-time growing graph embedding. To achieve the optimal graph embeddings over time, it optimizes the parameters of deep autoencoder for each snapshot through minimizing a weighted combination loss built by the embedding loss and the graph reconstruction loss, which can help to preserve both local and global graph structure properties. However, DynGEM only considers temporal patterns spanning in two consecutive time snapshots. Moreover, it assumes that the graph dynamic changes are smooth and use regularization terms to disallow rapid changes. To address these problems, Goyal *et al.* [25]

further proposed a dyngraph2vec model, including three variants (dyngraphAE, dyngraphRNN, and dyngraphAERNN) for learning dynamic graph series embeddings over long sequences of previous snapshots. The main goal of dyngraph2vec [25] is to predict the graph embedding for the future snapshot incrementally using the past snapshots. A recurrent neural network (RNN) layer was added to learn the temporal dynamics across different graph snapshots. Hence, dyngraph2vec [25] adopted a lookback parameter ($l = 1, 2$, and 3) to specify the number of lookback time steps for predicting the next snapshot. However, the model is large and computationally expensive.

D. GNN-Based Models

With the great success of graph neural networks (GNNs) in static graph representation learning, a few recent studies proposed time-dependent GNN models integrating GNN and RNN [or long short-term memory (LSTM)] for learning temporal graph sequence representations in different graph snapshots, e.g., TDGNN [26] and EvolveGCN [27]. More recently, Sankar *et al.* [28] and Rossi *et al.* [29] adopted the self-attention mechanism to jointly encode the structural and temporal dynamics for temporal graph sequences. The self-attention mechanism helps to effectively capture long-range dependencies and draw most relevant context from all past graph snapshots to adaptively assign interpretable weights for previous time steps. Moreover, there are also a few works modeling the dynamic graphs as temporal point processes in conjunction with the attention mechanism, e.g., DynamicTriad [30], HTNE [31], and DyRep [32]. More dynamic graph embedding literature can be found in the surveys [2], [33]–[35].

Nevertheless, the majority of existing dynamic graph embedding models are "deterministic," hence disregarding the important "uncertainty information" for the graph embeddings in the latent space. As we will demonstrate here, the uncertainty quantification is ultimately related to the embedding dimension of the system, with the effective dimension of the uncertainty being a lower bound for the optimum embedding dimension. This is a key finding of our work and is verified for all eight diverse benchmarks we consider in our current study.

III. METHODOLOGY

A. Notation

We model a dynamic graph as a series of T graph snapshots $\mathcal{G} = \{G_1, G_2, \ldots, G_T\}$, see an example in Fig. 1(a). The graph snapshot G_t at timestamp t consists of a vertex set $V_t = \{v_1, v_2, \ldots, v_{|\mathcal{V}_t|}\}$ of $|\mathcal{V}_t|$ nodes and an edge set $E_t = \{e_{i,j}|i, j \in |\mathcal{V}_t|\}$, where each edge $e_{i,j}$ in the graph connects two vertices v_i and v_j. A_t represents the corresponding adjacent matrix of graph snapshot G_t, which can be either weighted or unweighted, directed or undirected. The node features are denoted by $X_t \in \mathbb{R}^{\mathcal{V}_t \times D}$, where D is the dimensionality of the node attributes. The main goal of stochastic temporal graph embedding is to learn time-dependent graph mappings ($\mathcal{F} = \{f_1, f_2, \ldots, f_T\}$) for different graph snapshots

such that each graph node can be represented as a sequence of low-dimensional multivariate Gaussian distributions.

B. Problem Formulation

Here, motivated by the previous work "Graph-2-Gauss (G2G)" [37] that only focuses on learning graph Gaussian embeddings for static graphs, we propose an efficient and scalable graph Gaussian embedding framework ("DynG2G") for more complex time-evolving graphs. The main goal of our work is to learn temporal graph embeddings as low-dimensional statistical distributions by incorporating the important graph evolutionary information from the previous graph sequences. In particular, the proposed DynG2G is capable of projecting a temporal graph from a non-Euclidean space to low-dimensional time-dependent "function" space (unlike the conventional "deterministic vector" space introduced in most existing studies [24], [25], [27], [38]) via a series of stochastic mappings ($\mathcal{F}' = \{f_1', f_2', \ldots, f_T'\}$) as shown in Fig. 1(b). Each graph mapping f_t' transforms the graph snapshot (G_t) at timestamp t to a low-dimensional multivariate Gaussian distributions $H_t = \{h_t^i|i \in |\mathcal{V}_t|\}$, where each node can be represented as a joint normal distribution $h_t^i = \mathcal{N}(\mu_i, \Sigma_i)$ in terms of a mean vector $\mu_i \in \mathbb{R}^L$ and a covariance matrix $\Sigma_i \in \mathbb{R}^{L \times L}$ (L is the embedding size). Finally, both spatial graph structure properties and temporal graph dynamics can be maximally preserved in the latent space.

C. DynG2G: A Stochastic Temporal Graph Embedding Framework With Uncertainty Quantification

The main workflow of the DynG2G is shown in Fig. 1, involving four key phases: 1) generation of node triplet based on weighted k-hop sampling for every graph snapshot as shown in Fig. 1(a); 2) building a snapshot-level stochastic graph embedding model based on a simplified nonlinear feedforward neural network encoder (f') with one " hidden layer" followed by two separate projection layers (p_{mu} and p_{var}) for outputting the final low-dimensional multivariate Gaussian distributions, i.e., mean and variance vectors, respectively; 3) training recurrently the DynG2G model for the graph snapshot G_{t+1} at timestamp $t + 1$ by transferring the pretrained parameters from G_t at the previous timestamp t; and 4) optimization of the model with the node triplet-based contrastive ranking loss.

1) Generation of Node Triplet Based on k-Hop Neighboring Sampling Approach: In order to preserve graph topological properties in the latent space, we first sample the node neighbors based on different k ($k = 1, 2, 3, \ldots, K$) hops, e.g., N_{ik} denotes the k-hop node neighbors of node v_i. The K node hops ("node context") sampled for each node in the graph snapshot are then used to generate the corresponding node triplet set ($\mathcal{T}_t = \{(v_i, v_i^+, v_i^-)|v_i \in V_t\}$), where v_i is the anchor node and v_i^+ and v_i^- are its positive nodes and negative nodes, respectively. The shortest path (sp(.)) between positive node pair (v_i and v_i^+) and the shortest path between the negative node pairs (v_i and v_i^-) follow the constraint: $\mathrm{sp}(v_i, v_i^+) < sp(v_i, v_i^-)$. In other words, positive nodes (v_i^+)
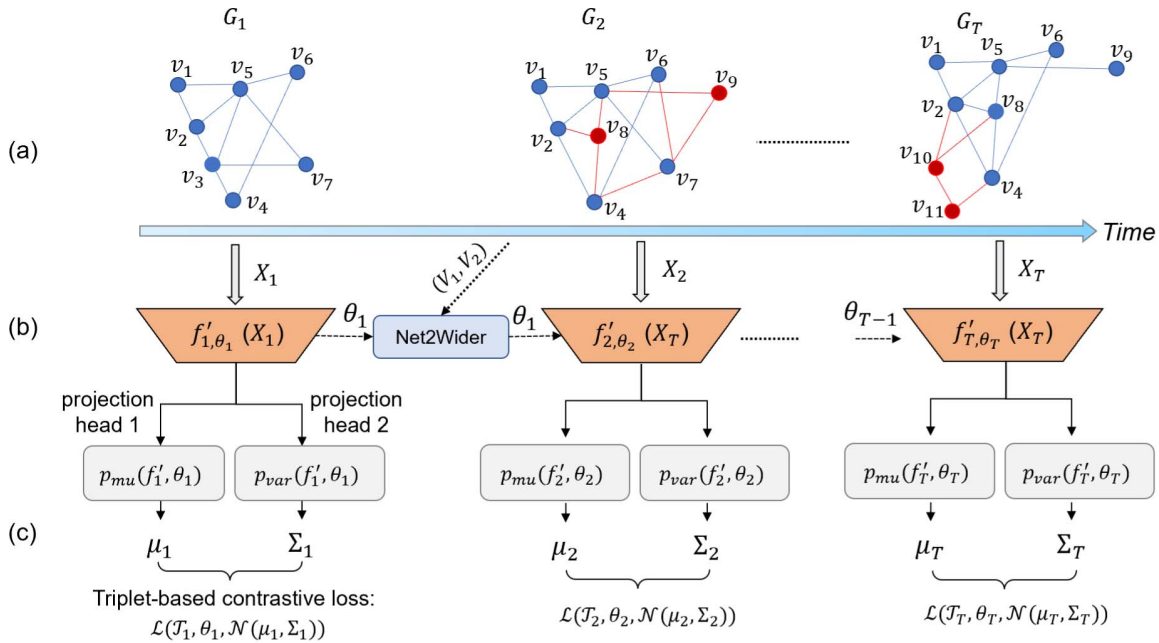
Fig. 1. Schematic of our proposed DynG2G framework for stochastic temporal graph embedding. (a) Dynamic graph G is composed of T graph snapshots. To learn the latent graph representations for $\mathcal{G} = \{G_1, G_2, \dots, G_T\}$ over T timestamps (vertices and links in red are the new added ones), we first generate for each node a set of node triplets ($\mathcal{T}_1$) based on the k-hop node neighbor sets computed from the adjacency matrix A_1 of G_1. (b) Attribute matrix X_1 ("one-hot encoding" of the nodes if a nonattributed graph) at the first timestamp is input to an encoder $f_1'(\cdot)$ with parameters θ_1 and two separate projection layers (p_{mu} and p_{var}), yielding the snapshot-level graph node embeddings in terms of mean vector $\mu_1^i \in \mathbb{R}^L$ and diagonal covariance matrix $\Sigma_1^i \in \mathbb{R}^{L \times L}$ for each node v_i (L is the embedding size). For the next snapshot training, we employ an extension of the Net2WiderNet approach [36] to adaptively expand the network hidden layer size based on the number of changing nodes in the next snapshot. (c) To effectively capture the temporal graph dynamics across different graph snapshots, we train the encoder $f_2(\cdot)'$ in the second timestamp with the hyperparameters (θ_2) transferred from the pretrained model for the first graph snapshot. We trained the model with a time-dependent node triplet-based contrastive loss $\mathcal{L}(\mathcal{T}_t, \theta_t, \mathcal{N}(\mu_t, \Sigma_t))$.

are much closer (i.e., within smaller hops) to the anchor node v_i than the negative nodes v_i^-. In order to achieve multiscale graph structure property preservation in the latent space, we ultimately rank the Kullback–Leibler (KL)-divergence dissimilarity between the latent node Gaussian embeddings over the sampled different K-hops for each node. The detailed node pair energy ranking formulas are shown in the following equation:

$$E(P_i, P_{k_1}) < E(P_i, P_{k_2}) < E(P_i, P_{k_3}) \cdots < E(P_i, P_{k_K})$$
$$\forall k_1 \in N_{i1}, k_2 \in N_{i2}, \dots, k_K \in N_{iK}. \quad (1)$$

2) Stochastic Temporal Contrastive Graph Embedding Learning: In comparison with the previous works that transform temporal graphs as deterministic "low-dimensional" vectors, in our work, we propose a stochastic temporal contrastive graph embedding learning method, which can transform each node of a graph snapshot into a density-based function space in terms of a multivariate Gaussian distribution via a nonlinear feedforward neural network (f_t' with one hidden layer of 512 units) and two separate projection heads (p_{mu} and p_{var}), which share the same parameters (θ_t) as the encoder f_t' to first learn the hidden representation for each graph node and then output the final stochastic graph node Gaussian embeddings ($\mathcal{N}_t(\mu_t, \Sigma_t)$) at timestamp t in terms of the mean vector ($\mu_t \in \mathbb{R}^L$) and the covariance matrix ($\Sigma_t \in \mathbb{R}^{L \times L}$) (the diagonal elements are the "variances"), with L the node embedding

size. More details about the snapshot-level encoding procedure can be seen from the first column of Fig. 1.

Subsequently, aiming to make our DynG2G model encode temporal graph patterns from the previous graph snapshot (G_{t-1}) to the next one G_t, we first input the weights to the Net2WiderNet model [36] and widen the encoder input layer size according to the increasing number of nodes in the G_t. Following [36], we assume that graph snapshots G_{t-1} and G_t have node numbers V_{t-1} and V_t, respectively, where $V_t > V_{t-1}$. After training the G2G encoder $f_{t-1}'(\cdot)$ for G_{t-1}, we need to widen the input layer size of the encoder $f_t'(\cdot)$ for the next timestamp based on the increased number of nodes ($V_t - V_{t-1}$); then, θ_{t-1} is replaced by θ_t. If G_{t-1} has V_{t-1} inputs and 512 outputs, and timestamp t has V_t inputs, then $\theta_{t-1} \in \mathbb{R}^{V_{t-1} \times 512}$ and $\theta_t \in \mathbb{R}^{V_t \times 512}$. Net2WiderNet enables the encoder to increase the layer size that has V_t inputs, with $V_t > V_{t-1}$. By defining a random mapping function $g : \{1, 2, \dots, V_t\} \mapsto \{1, 2, \dots, V_{t-1}\}$ that satisfies $g(i) = j$ if $j \leq V_{t-1}$ and else $g(i) = $ random sample from $\{1, 2, \dots, V_{t-1}\}$, then we define a new weight matrix U_{t-1} and U_t representing the weights for these new layers. Hence, the new weights in the next timestamp are obtained by

$$U_{t-1}^j = \theta_{t-1}^{g(j)}, \quad U_t^j = \frac{1}{|\{x | g(x) = g(j)\}|} \theta_t^{g(j)} \quad (2)$$

where the first V_{t-1} columns of $\theta^{(t-1)}$ are copied directly into U_{t-1}. Columns $V_{t-1} + 1$ through V_t of U_{t-1} are created by

choosing a random mapping as defined in g by replacement. Specifically, each column of θ_{t-1} may be copied multiple times. Regarding the weights in U_t, we could account for the multiplicity by dividing the weight by a factor given by $(1/|\{x|g(x) = g(j)\}|)$.

Therefore, the DynG2G architecture transfers the pretrained encoder's parameters in the previous timestamp for the initialization of the encoder's parameters at the current timestamp, which allows for more efficient and stable training and effectively captures the intersnapshot evolving patterns. Finally, with the obtained graph node Gaussian distribution embeddings in the latent space, we can build the objective function as a form of "triplet-based contrastive loss" [see (3)]. In the latent representation space, we use the KL-divergence measure to quantify the difference between two nodes' Gaussian distributions. The loss function contains two main terms: one is the positive pairs' energy term measuring the "KL-divergence" dissimilarity between the positive node pairs sampled from each graph snapshot; the other one is the negative pairs' energy term for measuring the "KL-divergence" dissimilarity between the negative node pairs. The detailed formulas are shown in (4). Note that a distinct property of our DynG2G method is that the obtained node embedding variance allows us to quantify important uncertainty information for temporal graph node embeddings in the latent space. Hence, our proposed DynG2G framework enables us to quantitatively analyze the time-dependent dynamics as well as the transition complexity of the dynamic system over time.

3) Model Optimization With Node Triplet-Based Contrastive Loss: In order to optimize the DynG2G model and achieve the optimal stochastic graph embeddings for dynamic graph $\mathcal{G}$, we implement the DynG2G model in PyTorch and apply the Adam optimizer to minimize the time-dependent "node triplet-based" contrastive loss such that the positive node pairs' energy (i.e., "KL-divergence") are minimized and the negative node pairs' embedding energy can be maximized. Hence, the positive node pairs' embeddings are more similar, while the negative pairs' embeddings are more dissimilar in the latent space

$$\mathcal{L}_t = \sum_{(v_i, v_i^+, v_i^-) \in \mathcal{T}_t} \left[\mathbb{E}^2_{(v_i, v_i^+)} + e^{-\mathbb{E}_{(v_i, v_i^-)}} \right] \tag{3}$$

where $\mathbb{E}_{(v_i, v_i^+)}$ and $\mathbb{E}_{(v_i, v_i^-)}$ refer to the KL-divergence [see (4)] between the multivariate Gaussian embeddings of positive node pairs and negative node pairs in the node triplet set ($\mathcal{T}_t$), respectively. $\mathcal{T}_t$ is generated by sampling each node's neighbors by k-hop neighborhood introduced in step 1) [37]

$$D_{KL}(\mathcal{N}_i(\mu^i, \Sigma^i) || \mathcal{N}_j(\mu^j, \Sigma^j))$$
$$= \frac{1}{2} \Big[\text{tr}(\Sigma^{j-1} \Sigma^i) + (\mu^j - \mu^i)^T \Sigma^{j-1} (\mu^j - \mu^i) - L$$
$$+ \log \frac{|\Sigma^j|}{|\Sigma^i|} \Big] \tag{4}$$

where $\text{tr}(\cdot)$ denotes the trace of a matrix and L corresponds to the dimensionality of the node's Gaussian embeddings in the latent space.

The details of the DynG2G algorithm are presented in Algorithm 1.

Algorithm 1: DynG2G

Input: Weights θ_{t-1} of the network at previous timestamp, Attribute matrix X_t, Number of vertices of current timestamp $\mathcal{V}_t$ and previous timestamp $\mathcal{V}_{t-1}$, node triplet set $\mathcal{T}_t$.

Output: Time-dependent probabilistic embeddings $H_t = \{\mathcal{N}(\mu_t^i, \Sigma_t^i)|i \in \mathbb{R}^{\mathcal{V}_t}\}$, where $\mu_t^i \in \mathbb{R}^{\mathcal{V}_t \times L}$, $\Sigma_t^i \in \mathbb{R}^{\mathcal{V}_t \times L \times L}$.

1 **for** $t \leftarrow 1 : T$ **do**
2   **for** *each epoch* **do**
3   Create a G2G Encoder model $f'_t(\cdot)$ with initialized weights θ_t
4   **if** $t = 1$ **then**
5   Initialize θ_t with random weights
6   **else**
7        $\theta_t = \theta_{t-1}$
8   **end**
9   **if** $\mathcal{V}_t > \mathcal{V}_{t-1}$ **then**
10        $f'_t(\cdot) \leftarrow Net2Wider(f'_{t-1}(\cdot))$
11   **end**
12      $\mu_t \leftarrow p_{mu}(f'_t, \theta_t)$
13      $\Sigma_t \leftarrow p_{var}(f'_t, \theta_t)$
14   Loss = $\mathcal{L}(\mathcal{T}_t, \theta_t, \mathcal{N}(\mu_t, \Sigma_t))$
15   Backpropagate the loss
16   **end**
17 **end**
18 **Return** H_t

We first train the encoder using the graph at the first timestamp G_1 with random initialization of model parameters θ_1. The subsequent timestamps are trained using the encoder with the condition that its initialization will be done using the parameters from the previous timestamp encoder θ_{t-1}. In order to handle the growing graphs, the encoder is "widened" if the number of nodes increases in the graphs of subsequent timestamps. The "widening" of encoder allows us to increase the number of neuron units in the input layer of the encoder while preserving the weights of the network from the earlier timestamps of the training. We handle the widening of the encoder by adopting and expanding the Net2WiderNet algorithm [36]. Once the network is trained for the $(t-1)$th timestamp, the network weights are first input to Net2WiderNet that widens the hidden layer size of the encoder on the basis of the number of "new nodes" [in red in Fig. 1(a)] added to graph snapshot. The output from the Net2WiderNet gives us the network weights to initialize the encoder at the next timestamp t. To this end, using the network parameters from the previous timestamp results in transferring temporal information from f'_{t-1} to f'_t effectively and achieves faster convergence of the training for subsequent timestamps.

IV. EXPERIMENTS AND ANALYSIS

A. Dataset Description

We evaluated our DynG2G model on eight different benchmarks with different temporal dynamics. The specific graph dataset statistics are shown in Table I. Fig. 2 shows
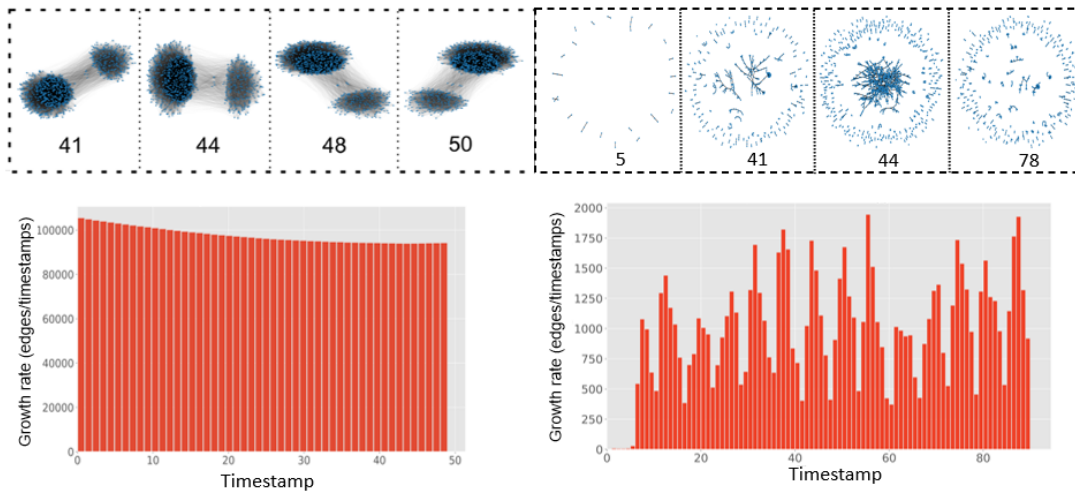
Fig. 2.   Visualization of the evolving dynamics of two temporal graphs for two of our eight benchmarks: synthetic SBM dataset (left) and Digg (right). Top row: snapshots of the two datasets at different timestamps (indicated by the number in each image). Bottom row: growth rate of edges as a function of time. SBM (small set) exhibits stable dynamics, whereas Digg (large set) exhibits rapidly changing dynamics. Similar visualizations of the dynamics of the other six benchmarks are shown in Fig. 5.

TABLE I

EXPERIMENT DATASET DESCRIPTION

Dataset	#Nodes	#Edges	#Timestamps	#Train/Val/Test
AS	6,474	13,895	100	70/10/20
SBM	1,000	4,870,863	50	35/5/10
Bitcoin-OTC	5,881	35,588	137	95/14/28
UCI	1,899	59,835	88	62/9/17
Slashdot	50,824	42,968	12	8/2/2
Facebook	46,873	857,815	30	21/3/6
Reality Mining	96	1,086,403	90	54/9/18
Digg	87,626	30,398	90	54/9/18

two temporal graph examples for the typical snapshots in the stochastic block model (SBM) and Digg datasets.

1) *Autonomous System (AS) Dataset*[3]*:* It consists of a communication network of who-talks-to-whom from the border gateway protocol (BGP) logs. The dataset can be used for predicting message exchange in the future. The AS dataset used in our experiment contains 6474 nodes and 13 895 edges with 100 timestamps in total.

2) *SBM Dataset*[4]*:* It is generated using the SBM model. The first snapshot of the dynamic graph is generated to have three equal-sized communities with in-block probability 0.2 and cross-block probability 0.01. To generate subsequent graphs, it randomly picks 10–20 nodes at each time step and moves them to another community. The final generated synthetic SBM graph contains 1000 nodes, 4 870 863 edges, and 50 timestamps.

3) *Bit-OTC Dataset*[5]*:* It is who-trusts-whom network of people who trade using Bitcoin on a platform called Bitcoin OTC. The Bit-OTC dataset contains 5881 nodes and 35 588 edges across 137 timestamps (weighted directed graph). This dataset exhibits highly transient dynamics.

4) *UCI Dataset*[6]*:* It contains sent messages between the users of the online student community at the University of California, Irvine. The UCI dataset contains 1899 nodes and 59 835 edges across 88 timestamps (directed graph). This dataset too exhibits highly transient dynamics.

5) *Slashdot Dataset*[7]*:* It is a large-scale social reply network for the technology website Slashdot. Nodes represent users and edges correspond to the replies of users. The edges are directed and start from the responding user. Edges are annotated with the timestamp of the reply. The Slashdot dataset contains 50 824 nodes and 42 968 edges across 12 timestamps.

6) *Facebook Dataset*[8]*:* It is a large-scale Facebook wall post dynamic network, where each node is a user and a temporal directed edge represents a post from one user on another user's wall at a given timestamp. The entire temporal graph contains 46 873 nodes and 857 815 edges over 30 timestamps sampled monthly from October 14, 2004 to January 21, 2009.

7) *Reality Mining Dataset*[9]*:* The network contains human contact data among 100 students of the Massachusetts Institute of Technology (MIT); the data were collected with 100 mobile phones over nine months in 2004. Each node represents a student; an edge denotes the physical contact between two nodes. In our experiment, the dataset contains 96 nodes and 1 086 403 undirected edges across 90 timestamps.

8) *Digg Dataset*[10]*:* It is a large-scale dynamic reply network of the social news website Digg. Nodes represent the users of the Digg website, while edges are used

[3]https://Snap.stanford.edu/Data/as-allstats.html

[4]https://Github.com/IBM/EvolveGCN/Tree/Master/data

[5]http://Snap.stanford.edu/Data/soc-sign-bitcoin-otc.html

[6]http://Konect.cc/Networks/opsahl-ucsocial/

[7]http://Konect.cc/Networks/slashdot-threads/

[8]https://Data.mendeley.com/Datasets/4dwzvcdsv3/2

[9]http://Realitycommons.media.mit.edu/realitymining.html

[10]http://Konect.cc/Networks/munmun_digg_reply/

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

XU *et al.*: DynG2G: EFFICIENT STOCHASTIC GRAPH EMBEDDING METHOD FOR TEMPORAL GRAPHS
7

to describe the replied message actions between users. It contains 87 626 nodes and 30 398 directed edges over 90 timestamps.

B. Performance Evaluation for Temporal Link Prediction

1) Evaluation Metrics: To evaluate our DynG2G model and demonstrate its effectiveness, we adopt two standard metrics.

1) *Mean Average Precision (MAP):* We rank all the predictions of node q in a decreasing order of the probabilities. Then, we calculate the average precision (AP) for each node with the formula shown in (5), where m is the number of connections predicted by our model for a particular node q, n is the maximum number of connections to be considered, $P(k)$ denotes the precision at the kth node, and $\text{rel}(k) = 1$ if kth node is connected to node q and otherwise 0. MAP can be computed by averaging the AP values of all the nodes Q (see the following equation):

$$\text{MAP} = \frac{1}{Q}\sum_{q}^{Q}\text{AP}(q)$$

$$\text{AP}(q) = \frac{1}{m}\sum_{k=1}^{n}[P(k) \times \text{rel}(k)]. \quad (5)$$

2) *Mean Reciprocal Rank (MRR):* To calculate the MRR, we rank all the predictions of node q in decreasing order of the probabilities, and then calculate the reciprocal rank $(1/k_q)$ of node q after finding the rank position (k_q) of the first relevant linked node for the query node q. Finally, we take the average reciprocal rank of all nodes as the MRR value, see the detailed formula in (6). Note that since MRR only considers the single highest ranked relevant node, MRR can give a general measure of quality for link prediction task; however, MAP considers whether all of the relevant nodes tend to get ranked highly

$$\text{MRR} = \frac{1}{Q}\sum_{q\in Q}\frac{1}{k_q}. \quad (6)$$

2) Implementation Details: Our implementation is based on the PyTorch framework, and all experiments were conducted on the NVIDIA Quadro RTX 6000 GPU (two 2.4-GHz 32 Core Processors; 1024-GB DDR4 3200-MHz Memory). We applied the Adam algorithm with hyperparameters (learning rate $= 1\text{e}^{-3}$, tolerance $= 100$, one hidden layer of size 512, and epochs $= 700$) to minimize the triplet-based square and exponential loss for different embedding sizes ($L = 16, 32, 64, 128,$ and 256). For the link prediction, the MLP model contains only one hidden layer with the same size as the node embedding size (L), and the learning rate is 1e^{-4}.

With the aforementioned two metrics, we evaluate our DynG2G model using the temporal link prediction task on eight benchmarks described in Section IV-A. The main purpose of temporal link prediction is to predict the links at timestamp $T + 1$ using the transformed graph sequence embeddings up to timestamp T. Specifically, we first utilize

the DynG2G model to obtain node embeddings (h_i^t) with k-hop neighbor factor ($K = 2$) for all timestamps of each benchmark. The timestamps in each benchmark were first split into train/validation/test by ratios of 70%, 10%, and 20% for link prediction, respectively; the specific timestamp numbers can be seen in Table I. Before training the MLP model for the link prediction, we performed "positive link" and "negative link" sampling similar to [37] for each timestamp in the training dataset. After that, each link feature was constructed as a 1-D vector, i.e., concatenate its nodes' embedding vectors as a single vector. Similarly, the validation and test link datasets were also built by the same procedure. Finally, we trained an MLP model with weighted cross-entropy loss for predicting the link probability in the test timestamps. We evaluate the temporal link prediction performance with different graph embedding sizes ($L \in [16, 32, 64, 128, 256]$).

We have carried out experiments for eight temporal benchmarks and performed comparisons with other baseline methods, including DynGEM [24], EvolveGCN [27], dyngraph2vecAE, and dyngraph2vecAERNN [25] when possible. We list the final results on temporal link predictions of MAP and MRR in Table II. The MAP values of DynG2G are based on the best means computed over five initializations and over time; the best means correspond to different embedding dimensions L, which here we call it the optimum embedding dimension L_o. For the MRR values, we report the values for that embedding dimension, which may not be necessarily the best values. DynG2G shows substantially better performance in terms of both metrics (MAP and MRR) for temporal link prediction compared to all other baseline methods, especially for the highly dynamic Bit-OTC dataset and UCI dataset. We note that for the UCI benchmark, the highest value obtained with DynG2G is $0.0347 = 0.0209 + 0.0138$, which is much higher than the second best 0.0270 obtained by EvolveGCN. For the last four benchmarks, there are no published results in the literature, so we could not make direct comparisons with respect to the accuracy. However, we managed to run EvolveGCN for two benchmarks only, as shown in Table II.

These eight benchmarks have different temporal dynamics whose complexity varies with time. In order to gain some insight into their dynamics, we plot the MAP metric for different embedding sizes L as a function of time for the testing period for one initialization for the SBM and Digg datasets in Fig. 3; many more detailed results are shown in Fig. 7, where we carry out five runs for each benchmark corresponding to different initializations. As we see, the value of MAP depends on the embedding dimension L and it is rather stable as a function of time for the SBM benchmark. However, for the highly transient Digg dataset, the MAP metric varies significantly in time, and different embedding sizes yield the best MAP value depending on the specific timestamp. The best embedding size, L, may vary for different initializations (see Fig. 7), so in Fig. 3 (bottom row), we present the MAP statistical results over five runs and over all timestamps. From these plots, we see that the best $L = 64$ for both the SBM and the Digg benchmarks, which is not possible to infer from a single initialization. Overall, this behavior is consistent with

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

8

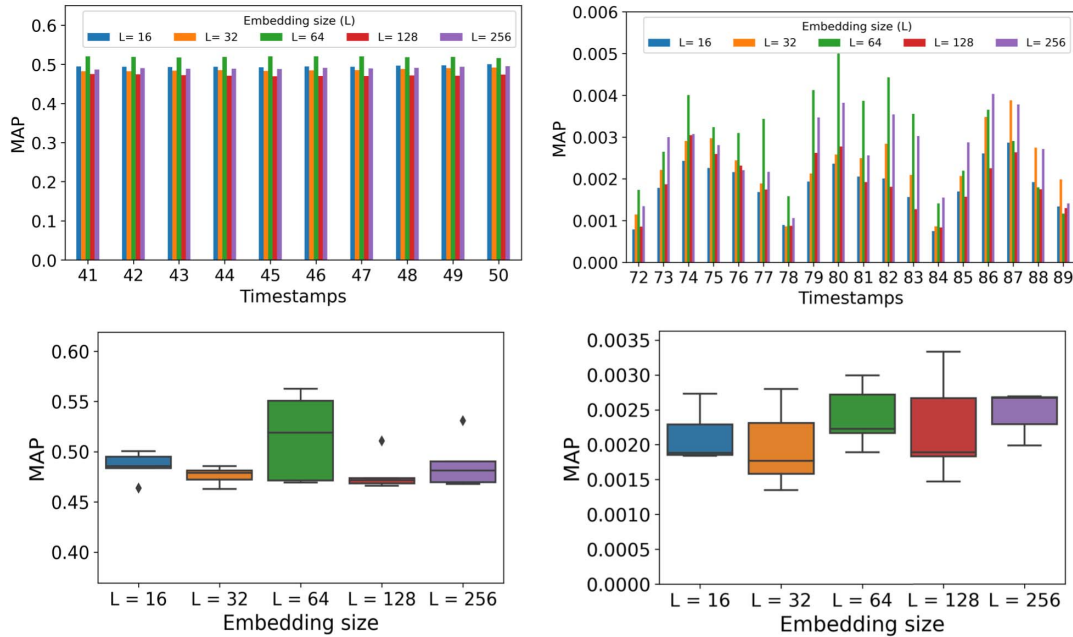IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS

Fig. 3.   MAP results for the temporal link prediction tasks with DynG2G (first row) on synthetic SBM dataset (left) and Digg dataset (right) for different embedding sizes ($L = 16, 32, 64, 128$, and 256). These results correspond to one specific initialization. In the second row, we show the MAP statistical results over five runs and over all timestamps. Results for the three other benchmarks are shown in Fig. 6, while details of the five initialization runs are shown in Fig. 7. The SBM graph evolves smoothly across different snapshots, and however, the Digg graph exhibits highly transitional dynamics across snapshots at different timestamps, as shown in Fig. 2.

TABLE II

COMPARISON RESULTS OF TEMPORAL LINK PREDICTION TASK FOR EIGHT DIFFERENT BENCHMARKS. DYNG2G MAP VALUES CORRESPOND TO THE BEST MEAN VALUES (AVERAGED OVER FIVE INTIALIZATIONS AND OVER TIME) SELECTED OVER ALL EMBEDDING DIMENSIONS. MRR VALUES CORRESPOND TO THE SAME EMBEDDING DIMENSION, WHICH WE CALL OPTIMUM L_o HERE, SEE TABLE III

Benchmark	Method	MAP	MRR
AS	DynGEM	0.0529	0.1028
	dyngraph2vecAE	0.0331	0.1028
	dyngraph2vecAERNN	0.0711	0.0493
	EvolveGCN	0.1534	0.3632
	DynG2G	**0.3154**	**0.3880**
SBM	DynGEM	0.168	0.0139
	dyngraph2vecAE	0.0983	0.0079
	dyngraph2vecAERNN	0.1593	0.012
	EvolveGCN	0.1989	0.0138
	DynG2G	**0.5146 ± 0.0435**	**0.0294 ± 0.0021**
Bit-OTC	DynGEM	0.0529	0.0921
	dyngraph2vecAE	0.0090	0.0916
	dyngraph2vecAERNN	0.0220	0.1268
	EvolveGCN	0.0028	0.0968
	DynG2G	**0.0662**	**0.4804**
UCI	DynGEM	0.0209	0.1055
	dyngraph2vecAE	0.0044	0.0540
	dyngraph2vecAERNN	0.0205	0.0713
	EvolveGCN	0.0270	0.1379
	DynG2G	**0.0209 ± 0.0138**	**0.1842 ± 0.0766**
Slashdot	**DynG2G**	**0.1637 ± 0.0227**	**0.4219 ± 0.0059**
Facebook	**DynG2G**	**0.1335 ± 0.0260**	**0.2967 ± 0.0216**
RealityM	**DynG2G**	**0.0266**	**0.0597**
	EvolveGCN	0.0090	0.0416
Digg	**DynG2G**	**0.0024 ± 0.0004**	**0.2210 ± 0.0380**
	EvolveGCN	0.0000126	0.0017

the insights provided by the uncertainty quantification analysis as we explain in Section IV-C.

C. Uncertainty Quantification for Node Embeddings

The existing temporal graph embedding methods are deterministic and they transform the temporal graph nodes as fixed points in the latent space, hence ignoring important information associated with node uncertainty. However, DynG2G is based on G2G static graph embedding method [37] and can provide uncertainty quantification, which is important for downstream analytics tasks. Specifically, G2G learns graph embeddings as multivariate Gaussian distributions, including the mean ("position") and variance ("uncertainty") vectors assigned to each node; the variance plays a key role in measuring the uncertainty of graph node representation. Moreover, as shown in [37] and also in [2], DynG2G can track the stable and unstable dimensions in time and hence obtain the effective dimensionality of the system, which is related to the required embedding size L. For temporally evolving graphs, this dimensionality may be changing in time, but DynG2G can capture in a quantified manner the transient dynamics of the evolving system. Hence, it is not surprising that the different datasets in the eight benchmarks exhibit different accuracies for different embedding sizes L as a function of time (e.g., Digg, Bit-OTC, and UCI benchmarks), whereas the benchmarks with smoother transient dynamics exhibit a more stable pattern in time with respect to L.

Correspondingly, the total variance as a function of time, plotted in Fig. 4, shows a similar dependence on the embedding size L. Interestingly, for the SBM benchmark, there is a clear separation of the values of variance for $L < 64$ but good convergence with respect to L for $L > 64$, and therefore, the effective dimensionality of the SBM system is $L = 64$, which does not change in time. The smooth and slow variation of
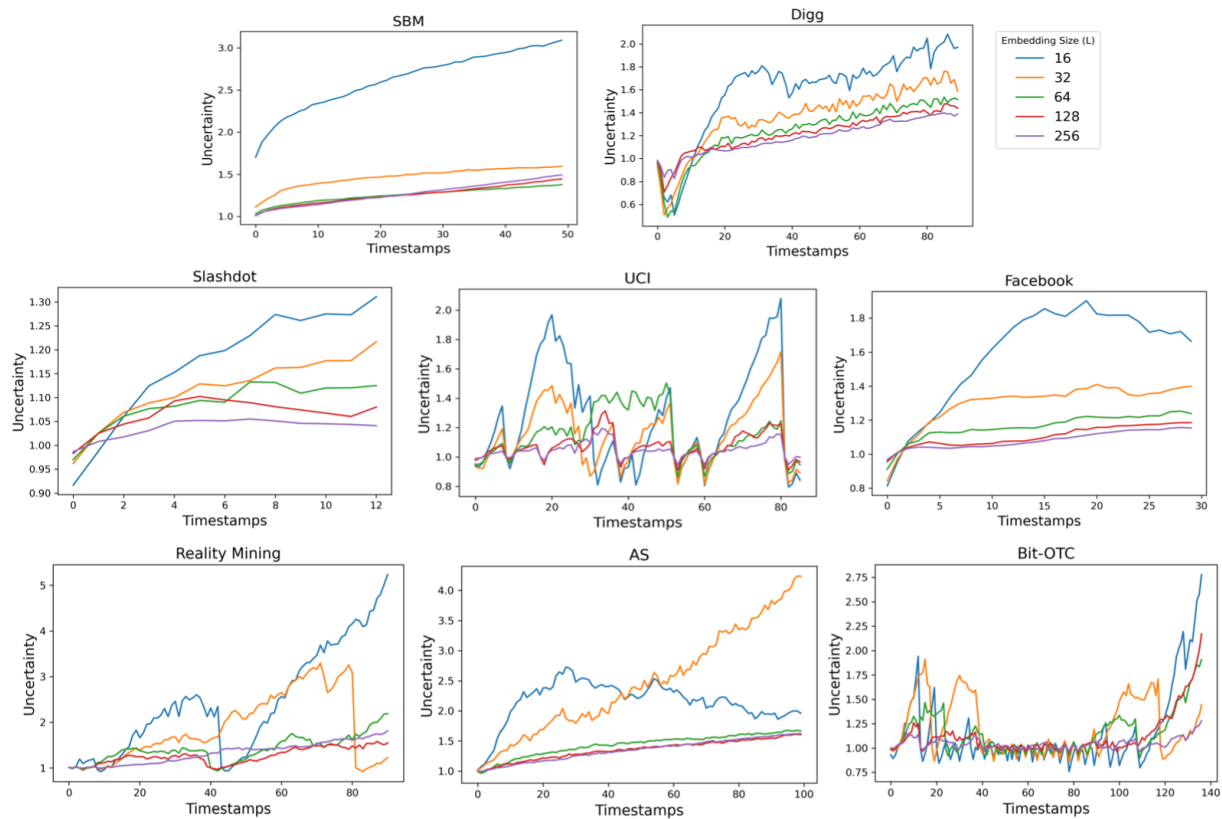
Fig. 4. Uncertainty versus time for temporal node embeddings with DynG2G for the eight benchmarks with embedding size ($L = 16, 32, 64, 128$, and 256). The X-axis represents the timestamps for each benchmark, while the Y-axis refers to the averaged standard deviations computed by the variances predicted by DynG2G for all nodes at different timestamps. These results correspond to a single initialization but more detailed results for all five initializations are shown in Fig. 8.

TABLE III
CORRELATION BETWEEN THE OPTIMAL EMBEDDING SIZE (L_o) AND THE EFFECTIVE DIMENSIONALITY OF UNCERTAINTY (D_u) FOR ALL EIGHT BENCHMARKS. THIS CORRELATION IS BASED ON AVERAGING THE RESULTS OVER ALL TIMESTAMPS AND OVER FIVE DIFFERENT RUNS CORRESPONDING TO DIFFERENT INITIALIZATIONS. THESE RESULTS SUGGEST A CLEAR PATH TO SELECTING THE GRAPH OPTIMUM EMBEDDING DIMENSION BY CHOOSING $L_o \geq D_u$

Benchmark	Optimal embedding size (L_o)	UQ dimensionality (D_u)
AS	64	64
SBM	64	64
UCI	256	256
Bit-OTC	256	256
Slashdot	64	64
Facebook	32	32
Reality Mining	64	64
Digg	64	64

uncertainty versus time for SBM is consistent with the stable results on MAP versus time shown in Fig. 3(a). We have observed similar trends for the AS benchmark, and here, the effective embedding size is $L = 64$ and does not change in time, consistent with the trends of the variance (uncertainty) versus time in Fig. 4. In contrast with the UCI and Bit-OTC benchmarks, we see that the variance exhibits a different

pattern and seems to converge for $L \geq 128$ for the UCI and $L = 256$ for the OTC. In fact, for the latter, the variance increases exponentially at longer times. This, in turn, reveals that both the UCI and Bit-OTC are high-dimensional systems with highly transient dynamics, so it is not surprising that the MAP accuracy varies also in time as a function of different embedding sizes. Finally, we note that the results presented in Fig. 4 are only for one initialization and a more systematic study is needed, as we present in the Appendix for different initializations (Fig. 8), to infer the proper value of the effective uncertainty dimensionality, which we denote as D_u, corresponding to the smallest uncertainty. It is interesting to note that the curves of uncertainty versus time look similar for different initialization runs as shown in the Appendix (Fig. 8), unlike the MAP versus times curves for different values of L.

V. CONCLUSION

We proposed an efficient stochastic dynamic graph embedding method (DynG2G), which enables us to automatically learn dynamic graph representations in low-dimensional "function" space with high efficiency. The learned graph embedding can be used for diverse downstream graph learning tasks, e.g., recommendation, fraud detection, and search. The current methods mostly rely on large autoencoder models (DynGEM and dyngraph2vecAE) or memory-based architectures that use RNN/LSTM for memory in addition to encoder–decoder

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

10   IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS
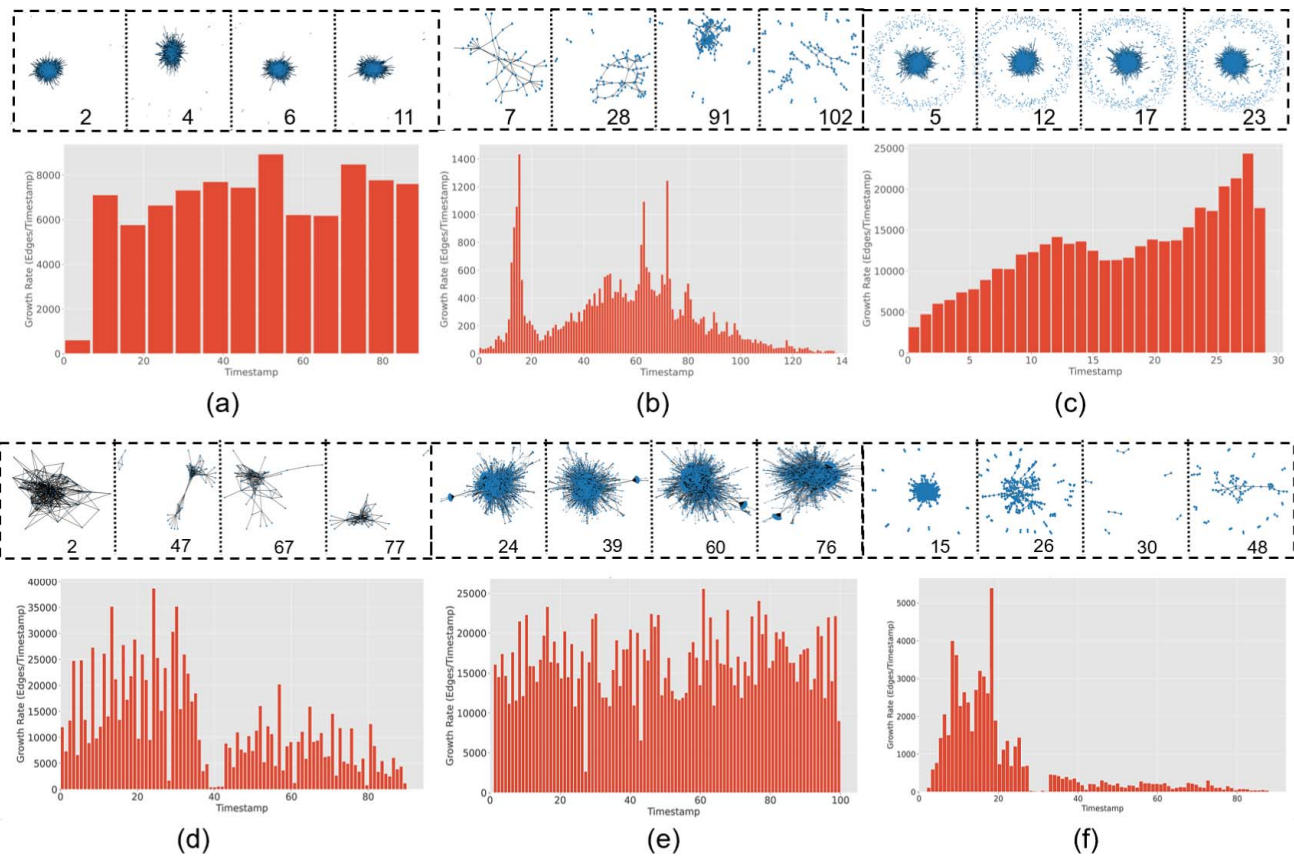

Fig. 5. Visualizations of the evolving dynamics of the remaining six temporal graphs in our benchmarks. (a) Slashdot, (b) Bit-OTC, (c) Facebook, (d) Reality Mining, (e) AS, and (f) UCI. In each subfigure, the top row shows four random snapshots of the temporal graph at different timestamps (indicated by the number in each image); the bottom row shows the growth rate of edges as a function of time in each benchmark. (a), (c), and (e) exhibit stable dynamics, whereas (b), (d), and (f) exhibit rapidly changing dynamics.
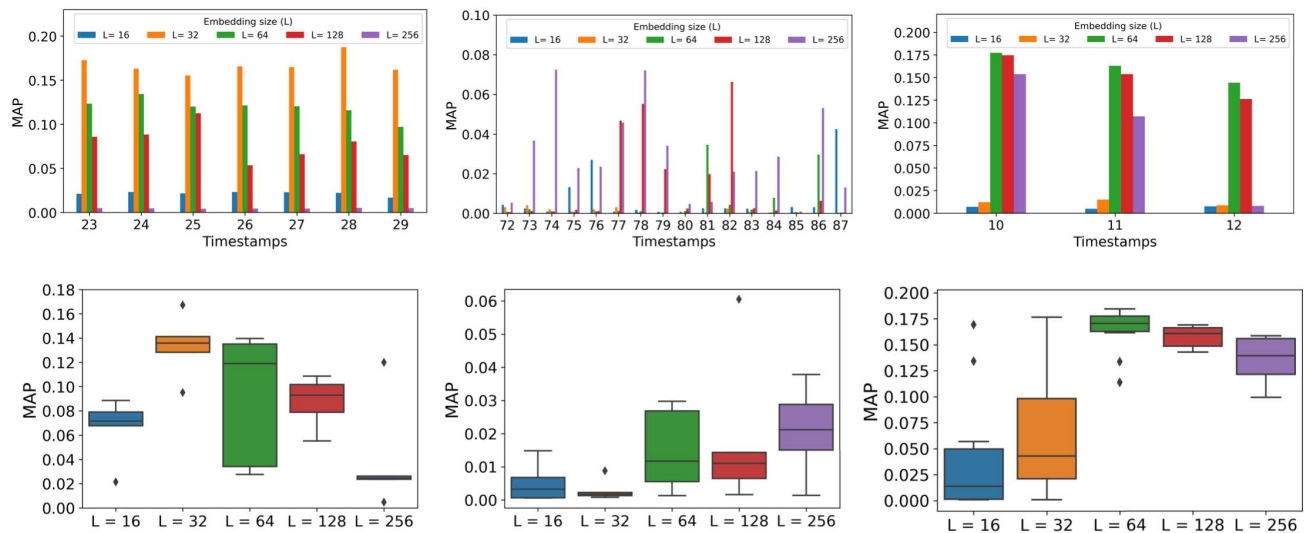
Fig. 6. MAP results for the temporal link prediction tasks with DynG2G for three other benchmarks (left column: Facebook, *middle column*: UCI, and right column: Slashdot) with different embedding size ($L = 16, 32, 64, 128,$ and 256). In the first row, we show the MAP results for the link prediction versus different timestamps for one initialization. In the second row, we show the MAP statistical results over five runs (corresponding to different initializations) and over all timestamps for Facebook, UCI, and Slashdot datasets.

networks (dyngraph2vecAERNN). EvolveGCN makes use of GCNs along with GRU and LSTM, and it consists of stacked layers of convolutional networks. DynG2G, in comparison, uses a single-layer neural network and at different time steps
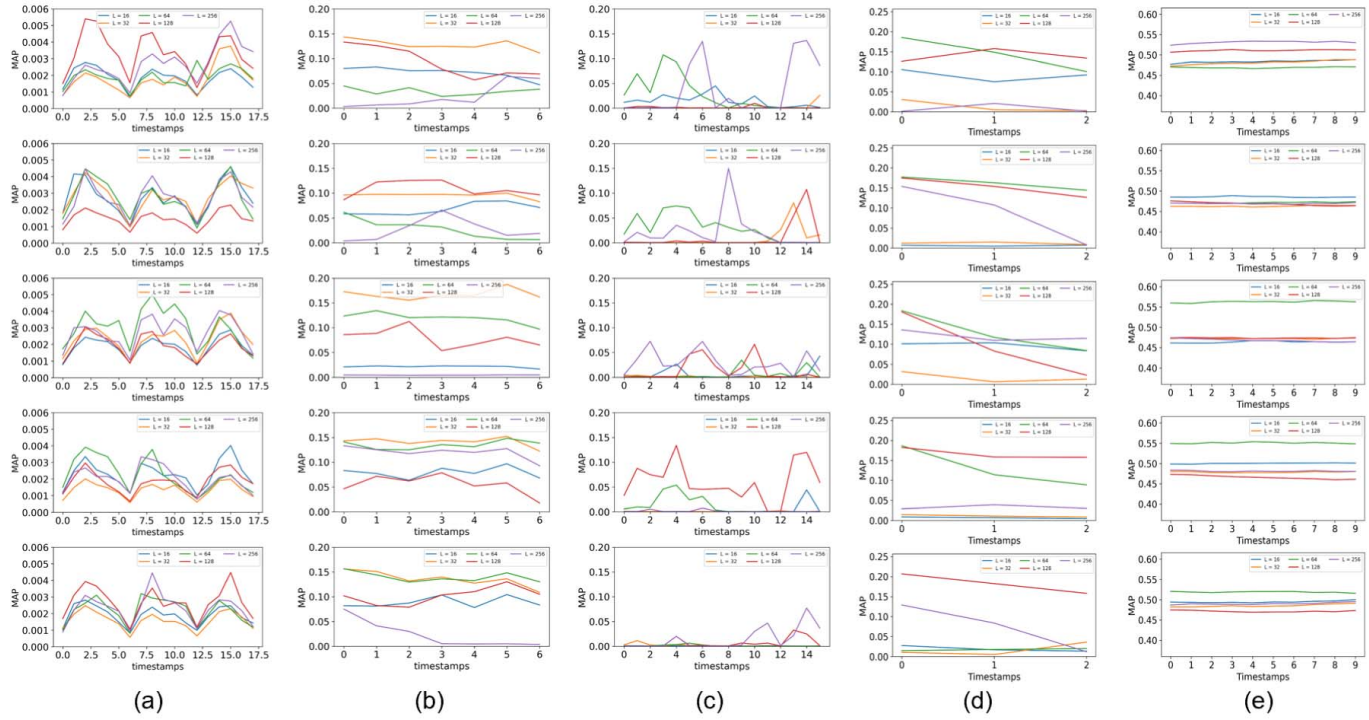
Fig. 7. MAP results using DynG2G method for temporal link prediction tasks for five different benchmarks with different embedding sizes ($L = 16, 32, 64, 128,$ and 256). (a)–(e) Digg, Facebook, UCI, Slashdot, and SBM benchmark, respectively. For each benchmark, we performed the temporal link prediction task with five different random initializations, as shown in the five different rows.
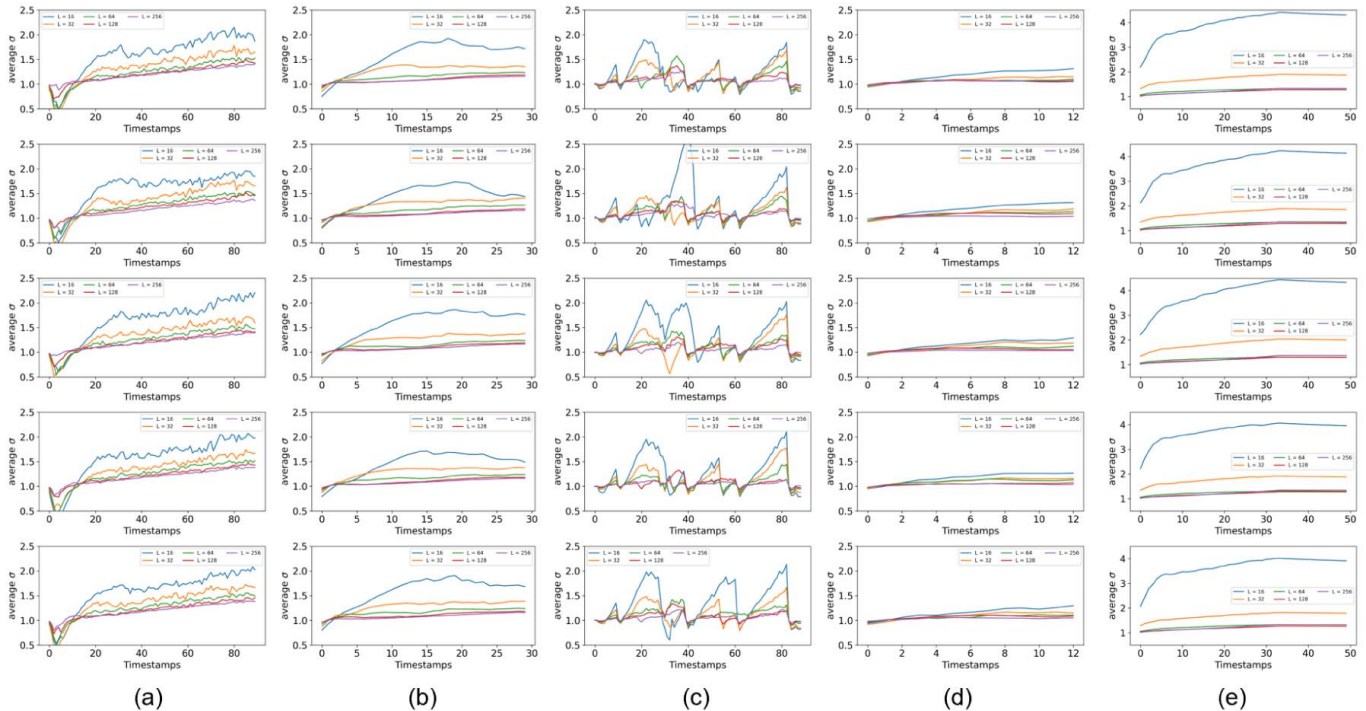

Fig. 8. Uncertainty (standard deviation) versus time for five runs (corresponding to different initializations) for temporal link prediction tasks with DynG2G for five different benchmarks with different embedding sizes ($L = 16, 32, 64, 128,$ and 256). (a)–(e) Digg, Facebook, UCI, Slashdot, and SBM benchmark, respectively. For each benchmark, we performed the temporal link prediction task with five different random initializations, as shown in the five different rows. The uncertainty is independent of the initialization and the variation is relatively smooth for each L except for the UCI benchmark that exhibits highly irregular dynamics.

of the graph, and we use the learned weights from the previous time step, which enables the model to converge faster by transferring temporal information.

We computed all eight benchmarks in about 0.6–60 s per epoch on a single GPU (see Table 1V for detailed computational costs of different embedding sizes and Fig. 10 for
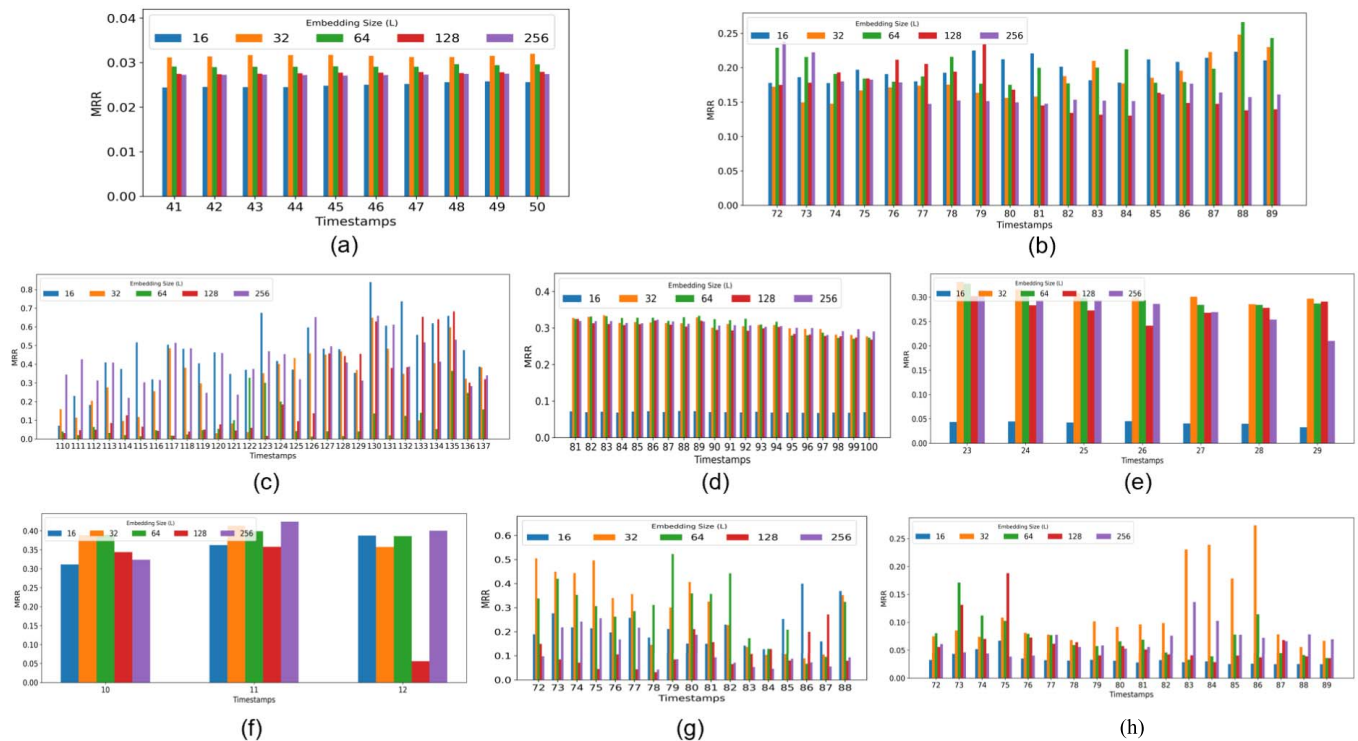
Fig. 9.   MRR results using DynG2G method for temporal link prediction tasks on eight different benchmarks: (a) SBM, (b) Digg, (c) Bit-OTC, (d) AS, (e) Facebook, (f) Slashdot, (g) UCI, and (h) Reality Mining.
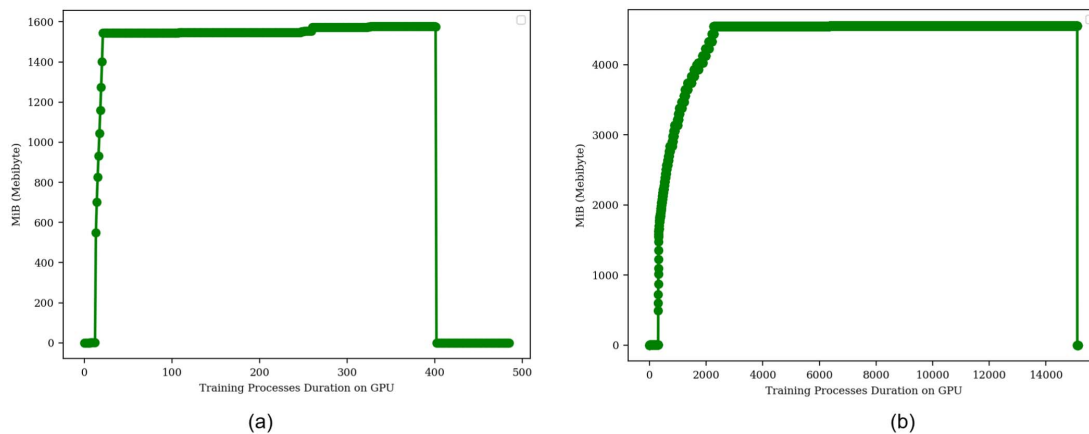

Fig. 10.   GPU memory footprint plots for the proposed DynG2G model trained with two different sizes of dynamic graph benchmarks (Digg and Reality Mining). (a) and (b) Amount of main memory that DynG2G code uses for learning graph embeddings for the Reality Mining dataset and Digg dataset, respectively. Reality Mining uses 1.6-GB maximum, while Digg uses ∼5-GB maximum, but it is persistent usage.

the GPU memory footprint for two of the benchmarks. Moreover, we computed the computational cost for the benchmark "Reality Mining" using EvolveGCN and we found that it takes about 3.40 s per epoch versus 0.698 s for DynG2G for embedding size $L = 16$. We obtained experimental results based on a range of graphs from 96 to 87 626 nodes with both slowly varying and rapidly changing dynamics in order to test DynG2G for diverse applications with the corresponding graphs characterized by multirate temporal dynamics. For example, we focused on the SBM and Digg benchmarks as the former has only 1000 nodes but close to 4.9 million edges, while the latter has 87 626 nodes and only 30 398 edges. Moreover, SBM exhibits stable dynamics,

whereas Digg exhibits rapidly evolving dynamics. The other six benchmarks shown in the Appendix have characteristics that fall between these two benchmarks. We have demonstrated that our DynG2G model outperforms other baseline methods (for available benchmark metrics) and effectively learns highly informative graph embeddings for even graphs with rapidly changing dynamics (i.e., Digg, Facebook, Reality Mining, Bit-OTC, and UCI benchmarks).

The novelty of our approach is that DynG2G in addition to the mean values; it can simultaneously predict the node embedding uncertainty, which plays a crucial role in quantifying the evolving uncertainty and complexity of dynamical systems over time. In particular, we obtained a

TABLE IV

AVERAGE AND STANDARD DEVIATION OF THE EPOCHWISE COMPUTATIONAL COST (IN SECONDS) CORRESPONDING TO DIFFERENT EMBEDDING SIZES ($L = 16, 32, 64, 128,$ AND 256) FOR DYNG2G OVER FIVE RUNS ON THE NVIDIA QUADRO RTX 6000 GPU (TWO 2.4-GHZ 32 CORE PROCESSORS AND 1024-GB DDR4 3200-MHZ MEMORY)

Benchmark	L = 16	L = 32	L = 64	L = 128	L = 256
SBM	9.363 ± 0.524	9.480 ± 0.402	9.228 ± 0.495	9.218 ± 0.334	8.960 ± 0.245
UCI	1.104 ± 0.010	1.076 ± 0.014	1.071 ± 0.011	1.075 ± 0.010	1.086 ± 0.010
Bit-OTC	3.533 ± 0.094	3.871 ± 0.063	3.818 ± 0.251	3.904 ± 0.717	3.806 ± 0.159
Slashdot	33.915 ± 0.133	33.768 ± 0.171	33.932 ± 0.232	34.203 ± 0.252	34.573 ± 0.281
Facebook	58.543 ± 10.321	58.687 ± 10.272	59.112 ± 10.251	59.517 ± 10.253	60.452 ± 10.271
Reality Mining	0.698 ± 0.069	0.668 ± 0.071	0.670 ± 0.071	0.672 ± 0.071	0.675 ± 0.071
Digg	23.465 ± 0.189	23.504 ± 0.233	23.632 ± 0.339	24.230 ± 1.190	26.232 ± 3.268

"universal" relation of the optimal embedding dimension, L_o, versus the effective dimensionality of uncertainty, D_u, and we found that $L_0 = D_u$ for all cases. To the best of our knowledge, this is the first time that such a relation is obtained for graph embedding of temporal graphs of arbitrary size, and it reveals that the uncertainty quantification approach we employ in the DynG2G algorithm correctly captures the intrinsic dimensionality of the dynamics of such evolving graphs despite the diverse nature and composition of the graphs at each timestamp.

A current limitation of DynG2G is that it employs a fixed embedding dimension, which is predefined by the user. However, based on our finding that a different embedding size is optimal at different timestamps, a new adaptive strategy combined with self-attention mechanism can be designed in future work to allow variable embedding size as the temporal dynamics of the systems evolves, especially for systems such as the Digg, UCI, and Bit-OTC benchmarks. Using the newly discovered $L_o - D_u$ correlation, we can develop a new criterion on how to select the optimum graph embedding dimension L_o on-the-fly by quantifying the effective dimensionality of corresponding node uncertainty D_u. We expect that this will lead to increasing both accuracy and efficiency of DynG2G. Another limitation of the model that is still very challenging to deal with is the missing timestamp data problem for real-world datasets, e.g., electronic healthcare record database. To this end, DynG2G has to be combined with state-of-the-art data imputation techniques in future work. Moreover, adopting a loss function such as the one from TransE [39] could extend our DynG2G to heterogeneous graphs. Finally, it will be interesting to develop a similar method but based on hyperbolic embeddings and examine whether the dimensionality reduction is even greater.

APPENDIX

We present additional results for all benchmarks to supplement the results in the main text in Figs. 5–8. We present results for the MRR metric in Fig. 9. In addition, we present the specific epochwise computational cost (in seconds) in Table IV corresponding to different embedding sizes ($L = 16, 32, 64, 128,$ and 256) for DynG2G over five runs.

ACKNOWLEDGMENT

REFERENCES

[1] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 4–24, Jan. 2020.

[2] M. Xu, "Understanding graph embedding methods and their applications," *SIAM Rev.*, vol. 63, no. 4, pp. 825–853, 2021.

[3] Y. Wang, C. Feng, L. Chen, H. Yin, C. Guo, and Y. Chu, "User identity linkage across social networks via linked heterogeneous network embedding," *World Wide Web*, vol. 22, no. 6, pp. 2611–2632, Nov. 2019.

[4] H. Ashoor *et al.*, "Graph embedding and unsupervised learning predict genomic sub-compartments from HiC chromatin interaction data," *Nature Commun.*, vol. 11, no. 1, pp. 1–11, Dec. 2020.

[5] M. Xu, Z. Wang, H. Zhang, D. Pantazis, H. Wang, and Q. Li, "A new graph Gaussian embedding method for analyzing the effects of cognitive training," *PLOS Comput. Biol.*, vol. 16, no. 9, Sep. 2020, Art. no. e1008186.

[6] M. Xu, D. L. Sanz, P. Garces, F. Maestu, Q. Li, and D. Pantazis, "A graph Gaussian embedding method for predicting Alzheimer's disease progression with MEG brain networks," *IEEE Trans. Biomed. Eng.*, vol. 68, no. 5, pp. 1579–1588, May 2021.

[7] E. Choi *et al.*, "Learning the graphical structure of electronic health records with graph convolutional transformer," in *Proc. AAAI Conf. Artif. Intell.*, vol. 34, no. 1, 2020, pp. 606–613.

[8] Y. Rong *et al.*, "Self-supervised graph transformer on large-scale molecular data," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 1–13.

[9] Y. Wang *et al.*, "Cell graph neural networks enable digital staging of tumour microenvironment and precisely predict patient survival in gastric cancer," *medRxiv*, Jan. 2021, doi: 10.1101/2021.09.01.21262086.

[10] J. Shlomi, P. Battaglia, and J.-R. Vlimant, "Graph neural networks in particle physics," *Mach. Learn., Sci. Technol.*, vol. 2, no. 2, Jan. 2021, Art. no. 021001.

[11] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013, *arXiv:1301.3781*.

[12] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 1225–1234.

[13] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016, *arXiv:1609.02907*.

[14] D. Zhu, C. Peng, Z. Zhang, P. Jian, and W. Zhu, "High-order proximity preserved embedding for dynamic networks," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 11, pp. 2134–2144, Nov. 2018.

[15] Y. Li, X. Wu, and A. Lu, "Analysis of spectral space properties of directed graphs using matrix perturbation theory with application in graph partition," in *Proc. IEEE Int. Conf. Data Mining*, Nov. 2015, pp. 847–852.

[16] Z. Zhang, P. Cui, J. Pei, X. Wang, and W. Zhu, "Timers: Error-bounded SVD restart on dynamic networks," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 224–231.

[17] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 855–864.

[18] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk: Online learning of social representations," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2014, pp. 701–710.

[19] U. Singer, I. Guy, and K. Radinsky, "Node embedding over temporal graphs," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, Aug. 2019, pp. 4605–4612.

[20] S. Mahdavi, S. Khoshraftar, and A. An, "Dynnode2vec: Scalable dynamic network embedding," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2018, pp. 3762–3765.

[21] L. Du, Y. Wang, G. Song, Z. Lu, and J. Wang, "Dynamic network embedding: An extended approach for skip-gram based network embedding," in *Proc. IJCAI*, 2018, pp. 2086–2092.

[22] W. Yu, W. Cheng, C. C. Aggarwal, K. Zhang, H. Chen, and W. Wang, "NetWalk: A flexible deep embedding approach for anomaly detection in dynamic networks," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2018, pp. 2672–2681.

[23] G. H. Nguyen, J. B. Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim, "Dynamic network embeddings: From random walks to temporal random walks," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2018, pp. 1085–1092.

[24] P. Goyal, N. Kamra, X. He, and Y. Liu, "DynGEM: Deep embedding method for dynamic graphs," 2018, *arXiv:1805.11273*.

[25] P. Goyal, S. R. Chhetri, and A. Canedo, "Dyngraph2vec: Capturing network dynamics using dynamic graph representation learning," *Knowl.-Based Syst.*, vol. 187, Jan. 2020, Art. no. 104816.

[26] L. Qu, H. Zhu, Q. Duan, and Y. Shi, "Continuous-time link prediction via temporal dependent graph neural network," in *Proc. Web Conf.*, Apr. 2020, pp. 3026–3032.

[27] A. Pareja, "Evolvegcn: Evolving graph convolutional networks for dynamic graphs," in *Proc. AAAI Conf. Artif. Intell.*, 2020, vol. 34, no. 4, pp. 5363–5370.

[28] A. Sankar, Y. Wu, L. Gou, W. Zhang, and H. Yang, "DySAT: Deep neural representation learning on dynamic graphs via self-attention networks," in *Proc. 13th Int. Conf. Web Search Data Mining*, Jan. 2020, pp. 519–527.

[29] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein, "Temporal graph networks for deep learning on dynamic graphs," 2020, *arXiv:2006.10637*.

[30] L. Zhou, Y. Yang, X. Ren, F. Wu, and Y. Zhuang, "Dynamic network embedding by modeling triadic closure process," in *Proc. AAAI Conf. Artif. Intell.*, vol. 32, no. 1, pp. 1–8, 2018.

[31] Y. Zuo, G. Liu, H. Lin, J. Guo, X. Hu, and J. Wu, "Embedding temporal network via neighborhood formation," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2018, pp. 2857–2866.

[32] R. Trivedi, M. Farajtabar, P. Biswal, and H. Zha, "Dyrep: Learning representations over dynamic graphs," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–25.

[33] P. Goyal, "Graph embedding algorithms for attributed and temporal graphs," *ACM SIGWEB Newslett.*, vol. 2020, pp. 1–4, Apr. 2020.

[34] S. Ji, S. Pan, E. Cambria, P. Marttinen, and P. S. Yu, "A survey on knowledge graphs: Representation, acquisition, and applications," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 2, pp. 494–514, Feb. 2022.

[35] Y. Xie, C. Li, B. Yu, C. Zhang, and Z. Tang, "A survey on dynamic network embedding," 2020, *arXiv:2006.08093*.

[36] T. Chen, I. Goodfellow, and J. Shlens, "Net2Net: Accelerating learning via knowledge transfer," 2015, *arXiv:1511.05641*.

[37] A. Bojchevski and S. Günnemann, "Deep Gaussian embedding of graphs: Unsupervised inductive learning via ranking," 2017, *arXiv:1707.03815*.

[38] B. Zhou, X. Zheng, Y. G. Wang, M. Li, and J. Gao, "Grassmann graph embedding," in *Proc. Workshop Geometrical Topological Represent. Learn.*, 2021, pp. 1–9.

[39] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 26, 2013, pp. 1–9.
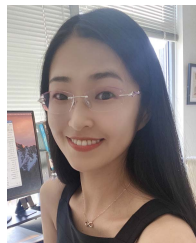
Mengjia Xu is currently a Post-Doctoral Associate with the McGovern Institute for Brain Research, Massachusetts Institute of Technology, Cambridge, MA, USA, and the Division of Applied Mathematics, Brown University, Providence, RI, USA. Her research interests include stochastic graph representation learning, optimization and generalization in deep neural networks, and multimodal biological data analysis.

Apoorva Vikram Singh received the B.S. degree from the Department of Electrical Engineering, National Institute of Technology, Silchar, India, in 2021. He is currently pursuing the M.Sc. degree in neural information processing with the University of Tübingen, Tübingen, Germany.

He is currently a Research Intern with the Division of Applied Mathematics, Brown University, Providence, RI, USA. His research interests include generalization in neural networks, graph neural networks, and computational neuroscience.

George Em Karniadakis is currently the Charles Pitts Robinson and John Palmer Barstow Professor of Applied Mathematics and Engineering at Brown University, Providence, RI, USA. He has a joint appointment with the Pacific Northwest National Laboratory (PNNL), Richland, WA, USA, where he is currently the Director of the PhILMs Collaboratory on physics-informed learning.