EvoNet: A Neural Network for Predicting the Evolution of Dynamic Graphs

Changmin Wu
LIX, École Polytechnique
changmin.wu@polytechnique.edu

Giannis Nikolentzos
LIX, École Polytechnique
nikolentzos@lix.polytechnique.fr

Michalis Vazirgiannis
LIX, École Polytechnique and AUEB
mvazirg@lix.polytechnique.fr

Abstract

Neural networks for structured data like graphs have been studied extensively in recent years. To date, the bulk of research activity has focused mainly on static graphs. However, most real-world networks are dynamic since their topology tends to change over time. Predicting the evolution of dynamic graphs is a task of high significance in the area of graph mining. Despite its practical importance, the task has not been explored in depth so far, mainly due to its challenging nature. In this paper, we propose a model that predicts the evolution of dynamic graphs. Specifically, we use a graph neural network along with a recurrent architecture to capture the temporal evolution patterns of dynamic graphs. Then, we employ a generative model which predicts the topology of the graph at the next time step and constructs a graph instance that corresponds to that topology. We evaluate the proposed model on several artificial datasets following common network evolving dynamics, as well as on real-world datasets. Results demonstrate the effectiveness of the proposed model.

1 Introduction

Graph neural networks (GNNs) have emerged in recent years as an effective tool for analyzing graph-structured data [1, 2, 3, 4]. These architectures bring the expressive power of deep learning into non-Euclidean data such as graphs, and have demonstrated convincing performance in several graph mining tasks, including graph classification [5], link prediction [6], and community detection [7, 8]. So far, GNNs have been mainly applied to tasks that involve static graphs. However, most real-world networks are dynamic, i. e., nodes and edges are added and removed over time. Despite the success of GNNs in various applications, it is still not clear if these models are useful for learning in dynamic scenarios. Although some models have been applied to this type of data, most studies have focused on predicting a low-dimensional representation (i. e., embedding) of the graph for the next time step [9, 10, 11, 12, 13, 14]. These representations can then be used in downstream tasks [9, 12, 15, 14]. However, predicting the topology of the graph (and not its low-dimensional representation) is a task that has not been properly addressed yet.

Graph generation, another important task in graph mining, has attracted a lot of attention from the deep learning community in recent years. The objective of this task is to generate graphs that exhibit specific properties, e. g., degree distribution, node triangle participation, community structure, etc. Traditionally, graphs are generated based on some network generation model such as the Erdős-Rényi model. These models focus on modeling one or more network properties, and neglect the others. Neural network approaches, on the other hand, can better capture the properties of graphs since they follow a supervised approach [16, 17, 18]. These architectures minimize a loss function such as the reconstruction error of the adjacency matrix or the value of a graph comparison algorithm.

Capitalizing on recent developments in neural networks for graph-structured data and graph generation, we propose in this paper, to the best of our knowledge, the first framework for predicting

the evolution of the topology of networks in time. The proposed framework can be viewed as an encoder-predictor-decoder architecture. The "encoder" network takes a sequence of graphs as input and uses a GNN to produce a low-dimensional representation for each one of these graphs. These representations capture structural information about the input graphs. Then, the "predictor" network employs a recurrent architecture which predicts a representation for the future instance of the graph. The "decoder" network corresponds to a graph generation model which utilizes the predicted representation, and generates the topology of the graph for the next time step. The proposed model is evaluated over a series of experiments on synthetic and real-world datasets, and is compared against several baseline methods. To measure the effectiveness of the proposed model and the baselines, the generated graphs need to be compared with the ground-truth graph instances using some graph comparison algorithm. To this end, we use the Weisfeiler-Lehman subtree kernel which scales to very large graphs and has achieved state-of-the-art results on many graph datasets [19]. Results show that the proposed model yields good performance, and in most cases, outperforms the competing methods.

The rest of this paper is organized as follows. Section 2 provides an overview of the related work and elaborates our contribution. Section 3 introduces some preliminary concepts and definitions related to the graph generation problem, followed by a detailed presentation of the components of the proposed model. Section 4 evaluates the proposed model on several tasks. Finally, Section 5 concludes.

2   RELATED WORK

Our work is related to random graph models. These models are very popular in graph theory and network science. The Erdős-Rényi model [20], the preferential attachment model [21], and the Kronecker graph model [22] are some typical examples of such models. To predict how a graph structure will evolve over time, the values of the parameters of these models can be estimated based on the corresponding values of the observed graph instances, and then the estimated values can be passed on to these models to generate graphs.

Other work along a similar direction includes neural network models which combine GNNs with RNNs [13, 23, 14]. These models use GNNs to extract features from a graph and RNNs for sequence learning from the extracted features. Other similar approaches do not use GNNs, but they instead perform random walks or employ deep autoencoders [11, 12]. All these works focus on predicting how the node representations or the graph representations will evolve over time. However, some applications require predicting the topology of the graph, and not just its low-dimensional representation. The proposed model constitutes the first step towards this objective.

3   EVONET: A NEURAL NETWORK FOR PREDICTING GRAPH EVOLUTION

In this Section, we first introduce basic concepts from graph theory, and define our notation. We then present EvoNet, the proposed framework for predicting the evolution of graphs. Since the proposed model comprises of several components, we describe all these components in detail.

3.1   PRELIMINARIES

Let $\mathcal{G} = (V, E)$ be an undirected, unweighted graph, where V is the set of nodes and E is the set of edges. We will denote by n the number of vertices and by m the number of edges. We define a permutation of the nodes of $\mathcal{G}$ as a bijective function $\pi : V \rightarrow V$, under which any graph property of G should be invariant. We are interested in the topology of a graph which is described by its adjacency matrix $A^\pi \in \mathbb{R}^{n \times n}$ with a specific ordering of the nodes π^1. Each entry of the adjacency matrix is defined as $A_{ij}^\pi = \mathbb{1}_{(\pi(v_i), \pi(v_j)) \in E}$ where $v_i, v_j \in V$. In what follows, we consider the "topology", "structure" and "adjacency matrix" of a graph equivalent to each other.

In many real-world networks, besides the adjacency matrix that encodes connectivity information, nodes and/or edges are annotated with a feature vector, which we denote as $X \in \mathbb{R}^{n \times d}$ and $L \in \mathbb{R}^{m \times d}$, respectively. Hence, a graph object can be also written in the form of a triplet $\mathcal{G} = (A, X, L)$. In this paper, we use this triplet to represent all graphs. If a graph does not contain node/edge

---

[1]For simplicity, the ordering π will be omitted in what follows.
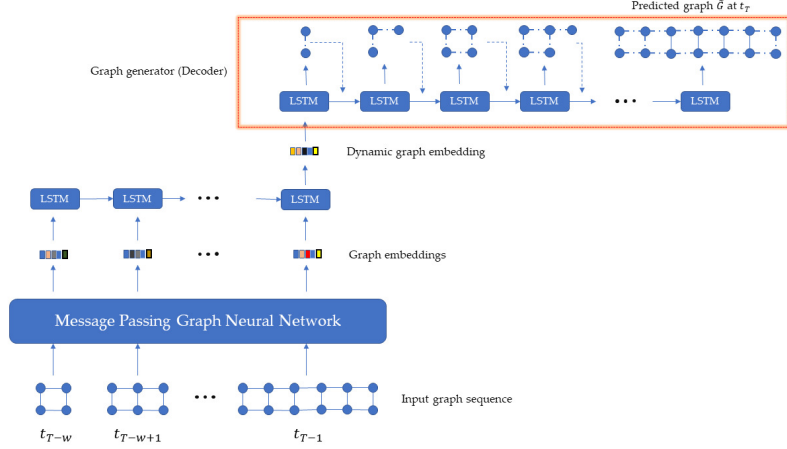
Figure 1: Illustration of the proposed architecture

attributes, we assign attributes to it based on local properties (e. g., degree, k-core number, number of triangles, etc).

An evolving network is a graph whose topology changes as a function of time. Interestingly, almost all real-world networks evolve over time by adding and removing nodes and/or edges. For instance, in social networks, people make and lose friends over time, while there are people who join the network and others who leave the network. An evolving graph is a sequence of graphs $\{\mathcal{G}_0, \mathcal{G}_1, \ldots, \mathcal{G}_T\}$ where $\mathcal{G}_t = (A_t, X_t, E_t)$ represents the state of the evolving graph at time step t. It should be noted that not only nodes and edges can evolve over time, but also node and edge attributes. However, in this paper, we keep node and edge attributes fixed, and we allow only the node and edge sets of the graphs to change as a function of time. The sequence can thus be written as $\{\mathcal{G}_t = (A_t, X, E)\}_{t \in [0,T]}$. We are often interested in predicting what "comes next" in a sequence, based on data encountered in previous time steps. In our setting, this is equivalent to predicting $\mathcal{G}_t$ based on the sequence $\{\mathcal{G}_k\}_{k<t}$. In sequential modeling, we usually do not take into account the whole sequence, but only those instances within a fixed small window of size w before $\mathcal{G}_t$, which we denote as $\{\mathcal{G}_{t-w}, \mathcal{G}_{t-w+1}, \ldots, \mathcal{G}_{t-1}\}$. We refer to these instances as the graph history. The problem is then to predict the topology of $\mathcal{G}_t$ given its history.

3.2 PROPOSED ARCHITECTURE

The proposed architecture is very similar to a typical sequence learning framework. The main difference lies in the fact that instead of vectors, in our setting, the elements of the sequence correspond to graphs. The combinatorial nature of graph-structured data increases the complexity of the problem and calls for more sophisticated architectures than the ones employed in traditional sequence learning tasks. Specifically, the proposed model consists of three components: (1) a graph neural network (GNN) which generates a vector representation for each graph instance, (2) a recurrent neural network (RNN) for sequential learning, and (3) a graph generation model for predicting the graph topology at the next time step. This framework can also be viewed as an encoder-predictor-decoder model. The first two components correspond to an encoder network which maps the sequence of graphs into a sequence of vectors and another network that predicts a representation for the next in the sequence graph. The decoder network consists of the last component of the model, and transforms the above representation into a graph. Figure 1 illustrates the proposed model. In what follows, we present the above three components of EvoNet.

3.2.1 ENCODING GRAPHS USING GRAPH NEURAL NETWORKS

Graph Neural Networks (GNNs) have recently emerged as a dominant paradigm for performing machine learning tasks on graphs. Several GNN variants have been proposed in the past years. All these models employ some message passing procedure to update node representations. Specifically,

each node updates its representation by aggregating the representations of its neighbors. After k iterations of the message passing procedure, each node obtains a feature vector which captures the structural information within its k-hop neighborhood. Then, GNNs compute a feature vector for the entire graph using some permutation invariant readout function such as summing the representations of all the nodes of the graph. As described below, the learning process can be divided into three phases: (1) aggregation, (2) update, and (3) readout.

Aggregation. In this phase, the network computes a message for each node of the graph. To compute that message for a node, the network aggregates the representations of its neighbors. Formally, at time $t + 1$, a message vector m_v^{t+1} is computed from the representations of the neighbors $\mathcal{N}(v)$ of v:

$$m_v^{t+1} = \text{AGGREGATE}^{t+1}\big(\big\{h_w^t \mid w \in \mathcal{N}(v)\big\}\big) \tag{1}$$

where AGGREGATE is a permutation invariant function. Furthermore, for the network to be end-to-end trainable, this function needs to be differentiable. In our implementation, AGGREGATE is a multi-layer perceptron (MLP) followed by a sum function.

Update. The new representation h_v^{t+1} of v is then computed by combining its current feature vector h_v^t with the message vector m_v^{t+1}:

$$h_v^{t+1} = \text{UPDATE}^{t+1}\big(h_v^t, m_v^{t+1}\big) \tag{2}$$

The UPDATE function also needs to be differentiable. To combine the two feature vectors (i.e., h_v^t and m_v^{t+1}), we employ the Gated Recurrent Unit proposed in [24]:

$$h_v^{t+1} = \text{GRU}^{t+1}(h_v^t, m_v^{t+1}) \tag{3}$$

Omitting biases for readability, we have:

$$
\begin{aligned}
r_v^{t+1} &= \sigma(W_R^{t+1} m_v^{t+1} + U_R^{t+1} h_v^t) \\
z_v^{t+1} &= \sigma(W_Z^{t+1} m_v^{t+1} + U_Z^{t+1} h_v^t) \\
\tilde{h}_v^{t+1} &= \tanh(W^{t+1} m_v^{t+1} + U^{t+1}(r_v^{t+1} \odot h_v^t)) \\
h_v^{t+1} &= (1 - z_v^{t+1}) \odot h_v^t + z_v^{t+1} \odot \tilde{h}_v^{t+1}
\end{aligned}
\tag{4}
$$

where the W and U matrices are trainable weight matrices, σ is the sigmoid function, and r_v and z_v are the parameters of the reset and update gates for a given node.

Readout. The *Aggregation* and *Update* steps are repeated for T time steps. The emerging node representations $\{h_v^T\}_{v \in V}$ are aggregated into a single vector which corresponds to the representation of the entire graph, as follows:

$$h_G = \text{READOUT}\big(\big\{h_v^T \mid v \in V\big\}\big) \tag{5}$$

where READOUT is a differentiable and permutation invariant function. This vector captures the topology of the input graph. To generate h_G, we utilize *Set2Set* [25]. Other functions such as the sum function were also considered, but were found less effective in preliminary experiments.

3.2.2 PREDICTING GRAPH REPRESENTATIONS USING RECURRENT NEURAL NETWORKS

Given an input sequence of graphs, we use the GNN described above to generate a vector representation for each graph in the sequence. Then, to process this sequence, we use a recurrent neural network (RNN). RNNs use their internal state (i.e., memory) to preserve sequential information. These networks exhibit temporal dynamic behavior, and can find correlations between sequential events. Specifically, an RNN processes the input sequence in a series of time steps (i.e., one for each element in the sequence). For a given time step t, the hidden state h_t at that time step is updated as:

$$h_{t+1} = f(h_t, x_{t+1}) \tag{6}$$

where f is a non-linear activation function. A generative RNN outputs a probability distribution over the next element of the sequence given its current state h_t. RNNs can be trained to predict the next element (e.g., graph) in the sequence, i.e., they can learn the conditional distribution

4

$p(G_t|G_1, \ldots, G_{t-1})$. In our implementation, we use a Long Short-Term Memory (LSTM) network that reads sequentially the vectors $\{h_{G_i}\}_{i \in [t-w, t-1]}$ produced by the GNN, and generates a vector h_{G_t} that represents the embedding of G_t. The embedding incorporates topological information and will serve as input to the graph generation module. The GNN component presented above can be seen as a form of an encoder network. It takes as input a sequence of graphs and projects them into a low-dimensional space. Then, this component takes the sequence of graph representations as input and predicts the representation of the graph at the next time step.

3.2.3 GRAPH GENERATION

To generate a graph that corresponds to the evolution of the current graph instance, we capitalize on a recently-proposed framework for learning generative models of graphs [16]. This framework models a graph in an autoregressive manner (i. e., a sequence of additions of new nodes and edges) to capture the complex joint probability of all nodes and edges in the graph. Formally, given a node ordering π, it considers a graph G as a sequence of vectors:

$$S_{\mathcal{G}}^{\pi} = (S_1^{\pi}, S_2^{\pi}, \ldots, S_{|V|}^{\pi}) \tag{7}$$

where $S_i^{\pi} = [a_{1,i}, \ldots, a_{i-1,i}] \in \{0,1\}^{i-1}$ is the adjacency vector between node $\pi(i)$ and the nodes preceding it ($\{\pi(1), \ldots, \pi(i-1)\}$). We adapt this framework to our supervised setting.

The objective of the generative model is to maximize the likelihood of the observed graphs of the training set. Since a graph can be expressed as a sequence of adjacency vectors (given a node ordering), we can consider instead the distribution $p(\hat{S}^{\pi}; \theta)$, which can be decomposed in an autoregressive manner into the following product:

$$
\begin{aligned}
p(\hat{S}^{\pi}; \theta) &= \prod_{i=1}^{|V|} p(\hat{S}_i^{\pi} | \hat{S}_{k:k<i}^{\pi}, \theta) \\
&= \prod_{i=1}^{|V|} \prod_{j=1}^{i-1} p(\hat{a}_{ji}^{\pi} | \hat{a}_{li:l<j}^{\pi}, \hat{S}_{k:k<i}^{\pi}, \theta)
\end{aligned}
\tag{8}
$$

This product can be parameterized by a neural network. Specifically, following [16], we use a hierarchical RNN consisting of two levels: (1) the graph-level RNN which maintains the state of the graph and generates new nodes and thus learns the distribution $p(\hat{S}_i^{\pi} | \hat{S}_{k:k<i}^{\pi})$ and (2) the edge-level RNN which generates links between each generated node and previously-generated nodes and thus learns the distribution $p(\hat{a}_{ji}^{\pi} | \hat{a}_{li:l<j}^{\pi})$. More formally, we have:

$$
\begin{aligned}
h_0 &= h_{\mathcal{G}_T} \\
h_i &= \text{RNN}_1(h_{i-1}, \hat{S}_{i-1}^{\pi}) \\
m_{0,i} &= h_i \\
m_{j,i} &= \text{RNN}_2(m_{j-1,i}, \hat{a}_{j-1,i}^{\pi}) \\
p(\hat{a}_{j,i}^{\pi} = 1) &= \sigma(m_{j,i}) \\
\hat{a}_{j,i}^{\pi} &\sim p
\end{aligned}
\tag{9}
$$

where h_i is the state vector of the graph-level RNN (i. e., RNN_1) that encodes the current state of the graph sequence and is initialized by $h_{\mathcal{G}_T}$, the predicted embedding of the graph at the next time step T. The output of the graph-level RNN corresponds to the initial state of the edge-level RNN (i. e., RNN_2). The resulting value is then squashed by a sigmoid function to produce the probability of existence of an edge $\hat{a}_{j,i}$. In other words, the model learns the probability distribution of the existence of edges and a graph can then be sampled from this distribution, which will serve as the predicted topology for the next time step T.

To train the model, the cross-entropy loss between existence of each edge and its probability of existence is minimized:

$$L = \sum_{i=1}^{|V|} \sum_{j=1}^{i-1} a_{j,i}^{\pi} \big(1 - p(\hat{a}_{j,i}^{\pi} = 1)\big) + (1 - a_{j,i}^{\pi}) p(\hat{a}_{j,i}^{\pi} = 1) \tag{10}$$

Node ordering. It should be mentioned that node ordering π has a large impact on the efficiency of the above generative model. Note that a good ordering can help us avoid the exploration of all possible node permutations in the sample space. Different strategies such as the Breadth-First-Search ordering scheme can be employed to improve scalability [16]. However, in our setting, the nodes are distinguishable, i. e., node v of G_i and node v of G_{i+1} correspond to the same entity. Hence, we can impose an ordering onto the nodes of the first instance of our sequence of graphs, and then utilize the same node ordering for the graphs of all subsequent time steps (we place new nodes at the end of the ordering).

4 EXPERIMENTS AND RESULTS

In this Section, we evaluate the performance of EvoNet on synthetic and real-world datasets for predicting the evolution of graph topology, and we compare it against several baseline methods.

4.1 DATASETS

We use both synthetic and real-world datasets. The synthetic datasets consist of sequences of graphs where there is a specific pattern on how each graph emerges from the previous graph instance, i. e., add/remove some graph structure at each time step. The real-world datasets correspond to single graphs whose nodes incorporate temporal information. We decompose these graphs into sequences of snapshots based on their timestamps. The size of the graphs in each sequence ranges from tens of nodes to several thousand of nodes.

Path graph. A path graph can be drawn such that all vertices and edges lie on a straight line. We denote a path graph of n nodes as P_n. In other words, the path graph P_n is a tree with two nodes of degree 1, and the other $n-2$ nodes of degree 2. We consider two scenarios. In both cases the first graph in the sequence is P_3. In the first scenario, at each time step, we add one new node to the previous graph instance and we also add an edge between the new node and the last according to the previous ordering node. The second scenario follows the same pattern, however, every three steps, instead of adding a new node, we remove the first according to the previous ordering node (along with its edge).

Cycle graph. A cycle graph C_n is a graph on n nodes containing a single cycle through all the nodes. Note that if we add an edge between the first and the last node of P_n, we obtain C_n. Similar to the above case, we use C_3 as the first graph in the sequence, and we again consider two scenarios. In the first scenario, at each time step, we increase the size of the cycle, i. e., from C_i, we obtain C_{i+1} by adding a new node and two edges, the first between the new node and the first according to the previous ordering node and the second between the new node and the last according to the previous ordering node. In the second scenario, every three steps, we remove the first according to the ordering node (along with its edges), and we add an edge between the second and the last according to the ordering nodes.

Ladder graph. The ladder graph L_n is a planar graph with $2n$ vertices and $3n-2$ edges. It is the cartesian product of two path graphs, as follows: $L_n = P_n \times P_2$. As the name indicates, the ladder graph L_n can be drawn as a ladder consisting of two rails and n rungs between them. We consider the following scenario: at each time step, we attach one rung (P_2) to the tail of the ladder (the two nodes of the rung are connected to the two last according to the ordering nodes).

For all graphs, we set the attribute of each node equal to its degree, while we set the attribute of all edges to the same value (e. g., to 1).

4.1.1 REAL-WORLD DATASETS

Besides synthetic datasets, we also evaluate EvoNet on six real-world datasets.[2] They can be divided into three groups based on the nature of their sources.

---

[2]All our datasets are publicly available through websites of [26] and [27].

	$|V|$	$|E|$	% Pos.Edges	Timespan	
				Begin	End
BTC-OTC	$5,881$	$35,592$	89%	2010-11-08	2016-01-25
BTC-Alpha	$3,783$	$24,186$	93%	2010-11-08	2016-01-22
UCI-Forum	899	$33,720$	—	2004-05-15	2004-10-26
UCI-Message	$1,899$	$59,835$	—	2004-04-15	2004-10-26
EU-Core	986	$332,334$	—	1970-01-01	1972-03-14
DNC	$1,891$	$39,264$	—	2013-09-16	2016-05-25

Table 1: Statistics of the 6 real-world datasets used in our experiments.

Bitcoin transaction networks. Contains graphs derived from the Bitcoin transaction network, a who-trust-whom network of people who trade using Bitcoin [28, 29]. Due to the anonymity of Bitcoin users, platforms seek to maintain a record of users' reputation in Bitcoin trades to avoid fraudulent transactions. The nodes of the network represent Bitcoin users, while an edge indicates that a trade has been executed between its two endpoint users. Each edge is annotated with an integer between -10 and 10, which indicates the rating of the one user given by the other user. The datasets are collected separately from two platforms: Bitcoin OTC and Bitcoin Alpha. For all graphs in these two datasets, we set the attribute of each node equal to the average rating that the user has received from the rest of the community, and the attribute of each edge equal to the rating between its two endpoint users.

Social networks. Contains graphs generated from an online social network at the University of California, Irvine [30, 31]. It has two datasets: one is derived from the private message exchange between users; the other is based on the same user community, but focuses on their activity in the forum, i.e., public comment on a specific topic. The nodes of the networks represent users and the edges represent a message exchange or a shared interest (on a topic). All graphs in these two datasets are unweighted and unlabeled, thus we simply set the attribute of each node equal to its degree.

Email exchange networks. Contains two datasets derived from two sources. The first is generated using email data from a large European research institution [32], i.e., all incoming and outgoing email between members of the research institution. The second is collected from the 2016 Democratic National Committee (DNC) email leak [27], where the links denote email exchanges between DNC members. Similar to social network datasets, the graphs in these two datasets are also unweighted and unlabelled, thus we treat them the same way.

More details about these datasets are given in Table 1.

4.2 BASELINES

We compare EvoNet against several random graph models: (1) the Erdős-Rényi model [20], (2) the Stochastic Block model [33, 34], (3) the Barabási–Albert model [21], and (4) the Kronecker Graph model [22]. These are the traditional methods to study the topology evolution of temporal graphs, by proposing a driven mechanism behind the evolution. To be precise, these models begin with an initial graph and a rule to connect new emerged nodes with existing ones, then gradually grow the initial graph to the expected size following this rule, i.e., in Barabási–Albert model, we begin with a triangle and follow the preferential attachment rule, in which the probability of having an edge between a newly added node and an existing one is proportional to the current degree of the existing node.

4.3 EVALUATION METRIC AND EVALUATION SETUP

4.3.1 SYNTHETIC DATASETS

In general, it is very challenging to measure the performance of a graph generative model since it requires comparing two graphs to each other, a long-standing problem in mathematics and computer science [35]. We propose to use graph kernels to compare graphs to each other, and thus to evaluate the quality of the generated graphs. Graph kernels have emerged as one of the most effective tools for graph comparison in recent years [36]. A graph kernel is a symmetric positive semidefinite function
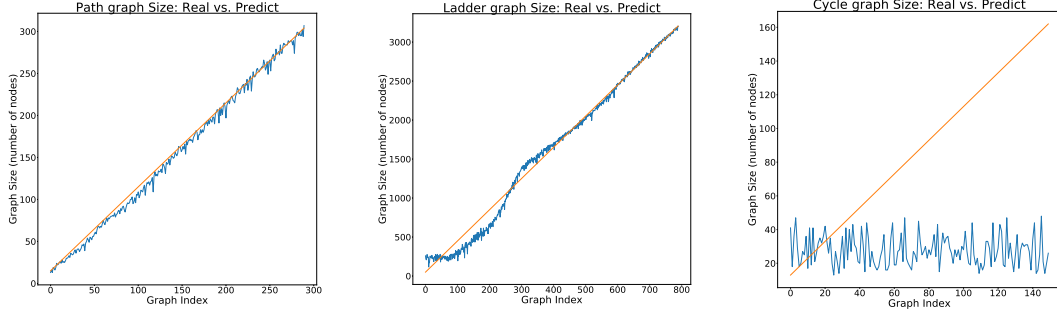
Figure 2: Results of synthetic datasets. Comparison of graph size (path, ladder and cycle graphs from left to right): predicted size (blue) VS. real size (orange).
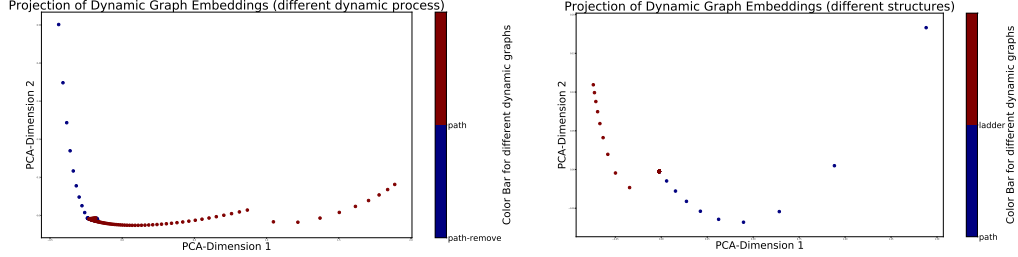

Figure 3: 2D projection of dynamic embeddings learned from datasets with different structures or different dynamics

which takes two graphs as input, and measures their similarity. In our experiments, we employ the Weisfeiler-Lehman subtree kernel which counts label-based subtree-patterns [19]. Note that we also normalize the kernel values, and thus the emerging values lie between 0 and 1.

As previously mentioned, each dataset corresponds to a sequence of graphs where each sequence represents the evolution of the topology of a single graph in 1000 time steps. We use the first 80% of these graph instances for training and the rest of them serve as our test set. The window size w is set equal to 10, which means that we feed 10 consecutive graph instances to the model and predict the topology of the instance that directly follows the last of these 10 input instances. Each graph of the test set along with its corresponding predicted graph is then passed on to the Weisfeiler-Lehman subtree kernel which measures their similarity and thus the performance of the model.

The hyperparameters of EvoNet are chosen based on its performance on a validation set. The parameters of the random graph models are set under the principle that the generated graphs need to share similar properties with the ground-truth graphs. For instance, in the case of the Erdős-Rényi model, the probability of adding an edge between two nodes is set to some value such that the density of the generated graph is identical to that of the ground-truth graph. However, since the model should not have access to such information (e. g., density of the ground-truth graph), we use an MLP to predict this property based on past data (i. e., the number of nodes and edges of the previous graph instances). This is in par with how the proposed model computes the size of the graphs to be generated (i. e., using also an MLP).

4.4 RESULTS

We next present the experimental results and compare the performance of EvoNet against that of the baselines.

Synthetic datasets. Figure 2 illustrates the experimental results on the synthetic datasets. Since the graph structures contained in the synthetic datasets are fairly simple, it is easy for the model to generate graphs very similar to the ground-truth graphs (normalized kernel values > 0.9). Hence, instead of reporting the kernel values, we compare the size of the predicted graphs against that of the
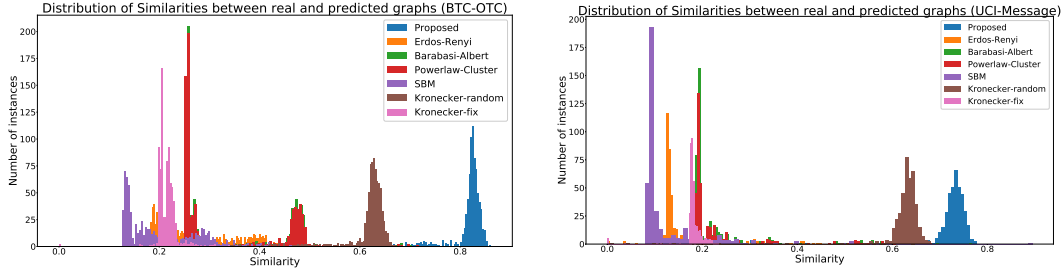
Figure 4: Similarity histograms on BTC-OTC (left) and UCI-Message (right) datasets. Blue one is the result of EvoNet, which is compared against 6 random graph models.

Stat. Model	BTC-OTC		BTC-ALPHA		UCI-Forum		UCI-Mesg		EU-Core		DNC	
	Mean	90%ile	Mean	90%ile	Mean	90%ile	Mean	90%ile	Mean	90%ile	Mean	90%ile
ER	0.28	0.40	0.22	0.32	0.15	0.16	0.16	0.26	0.06	0.09	0.82	1.00
SBM	0.21	0.30	0.18	0.27	0.10	0.10	0.13	0.22	0.03	0.05	0.84	1.00
BA	0.35	0.48	0.23	0.28	0.29	0.31	0.23	0.33	0.12	0.16	**0.85**	1.00
Power	0.35	0.48	0.23	0.28	0.29	0.31	0.23	0.33	0.12	0.16	0.85	1.00
Kron-Rand	0.62	0.64	0.44	0.47	0.59	0.62	0.62	0.65	0.60	0.65	0.01	0.02
Kron-Fix	0.21	0.23	0.08	0.11	0.44	0.47	0.18	0.20	0.53	0.55	0.01	0.06
EvoNet	**0.82**	**0.84**	**0.55**	**0.59**	**0.64**	**0.68**	**0.71**	**0.75**	**0.76**	**0.81**	0.83	**1.00**

Table 2: Statistics on the similarity distribution of different models: ER stands for Erdős–Rényi model. SBM stands for Stochastic Block Model. BA is Barabási–Albert Model. POWER is another model, similar to the Barabási–Albert, that grows graphs with powerlaw degree distribution. Kron-Rand represents the Kronecker Graph Model with learnable parameter while Kron-Fix represents the Kronecker Graph Model with fixed parameters.

ground-truth graphs. The figures visualize the increase of graph size on real sequence (orange) and predicted sequence (blue). For path graphs, in spite of small variance, we have an accurate prediction on the graph size. For ladder graph, we observe a mismatch at the beginning of the sequence for small size graphs but then a coincidence of the two lines on large size graphs. This mismatch on small graphs may be due to a more complex structure in ladder graphs such as cycles, as supported by the results of cycle graph on the right figure, where we completely mispredict the size of cycle graphs. In fact, we fail to reconstruct the cycle structure in the prediction, with all the predicted graphs being path graphs. This failure could be related to the limitations of GNN mentioned in [37].

Dynamic graph embedding. It is also important to check whether, in our encode-decoder framework, the learned code, which we refer to as "dynamic graph embedding", is really meaningful.[3] We design two experiments to verify the effectiveness of our embedding, with the help of synthetic graphs. In the first experiment, we take as input two sequences of graphs belonging to the same class but following different evolution dynamics. Specifically we took path graph and path graph with removal. In the second experiment, we control the evolution dynamic and vary the structures of graphs, where we use path graphs and ladder graphs following the same evolution of increasing size. The dynamic graph embeddings of different datasets learned from these experiments are recorded and visualized in Figure 3. Each point represents the projections of embeddings of each graph in the sequence into a 2-dimensional space by Principle Component Analysis (PCA). As we can see from the figure, embeddings learned from different datasets, either with different dynamics or with different structure, are both well separated, which suggests that the embeddings are meaningful, and those from the same dataset form special patterns such as a line in the space, which suggests a temporal dependency between these embeddings as they are learned from sequential data.

Real-World datasets. Finally, we analyze the performance of our model on the six real datasets. We obtain the similarities between each pair of real and predicted graphs in the sequence and draw a histogram to illustrate the distribution of similarities. Due to page limit, we choose to only show the histogram plot of BTC-OTC and UCI-Message datasets in Figure 4. Among all the traditional

---

[3]By "meaningful", we mean that the code(embedding) captures both structural feature of the graph class and temporal evolution of the series. Thus it can be applied to predict the graph at the future timestep.

9

random graph models, Kronecker graph model (with learnable parameter) performs the best, however on both datasets, our proposed method EvoNet (in blue) outperforms tremendously all other methods, with an average similarity of 0.82 on BTC-OTC dataset and 0.71 on UCI-Message dataset. Detailed statistics and results for other datasets can be found in Table 2, where we can find that our proposed model performs consistently better than the traditional methods.[4]

Overall, despite a failure in capturing some specific structures discovered in synthetic datasets, our experiments demonstrate the advantage of EvoNet over the traditional random graph models on predicting the evolution of dynamic graphs, especially for real world data with complex structures.

5   CONCLUSION

In this paper, we proposed EvoNet, a model that predicts the evolution of dynamic graphs, following an encoder-decoder framework. The proposed model consists of three components: (1) a graph neural network which transforms graphs to vectors, (2) a recurrent architecture which reads the input sequence of graph embeddings and predicts the embedding of the graph at the next time step, and (3) a graph generation model which takes this embedding as input and predicts the topology of the graph. We also proposed an evaluation methodology for this task which capitalizes on the well-established family of graph kernels. We apply the above methodology to demonstrate the predictive power EvoNet. Experiments show that the proposed model outperforms traditional random graph methods on both synthetic and real-world datasets. We should note that there is still space for improvement. Improving the efficiency of the proposed model and its scalability on large graphs are potential directions for future work.

REFERENCES

[1] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[2] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017.

[3] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

[4] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

[5] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.

[6] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*, pages 5165–5175, 2018.

[7] Joan Bruna and X Li. Community detection with graph neural networks. *stat*, 1050:27, 2017.

[8] Zhengdao Chen, Xiang Li, and Joan Bruna. Supervised community detection with line graph neural networks. *arXiv preprint arXiv:1705.08415*, 2017.

[9] Cheng Li, Xiaoxiao Guo, and Qiaozhu Mei. Deepgraph: Graph structure predicts network growth. *arXiv preprint arXiv:1610.06251*, 2016.

[10] Jundong Li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. Attributed network embedding for learning in a dynamic environment. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 387–396. ACM, 2017.

---

[4]Interested readers are kindly referred to supplemental materials for a full illustration of the results on all datasets.

[11] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. Continuous-time dynamic network embeddings. In *Companion Proceedings of the The Web Conference 2018*, pages 969–976. International World Wide Web Conferences Steering Committee, 2018.

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

[13] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. Structured sequence modeling with graph convolutional recurrent networks. In *International Conference on Neural Information Processing*, pages 362–373. Springer, 2018.

[14] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kaneza-shi, Tim Kaler, and Charles E Leisersen. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. *arXiv preprint arXiv:1902.10191*, 2019.

[15] Changping Meng, S Chandra Mouli, Bruno Ribeiro, and Jennifer Neville. Subgraph pattern neural networks for high-order graph evolution prediction. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

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

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

[18] Aditya Grover, Aaron Zweig, and Stefano Ermon. Graphite: Iterative generative modeling of graphs. *arXiv preprint arXiv:1803.10459*, 2018.

[19] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep): 2539–2561, 2011.

[20] Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[21] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

[22] Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin Ghahra-mani. Kronecker graphs: An approach to modeling networks. *Journal of Machine Learning Research*, 11(Feb):985–1042, 2010.

[23] Franco Manessi, Alessandro Rozza, and Mario Manzo. Dynamic graph convolutional networks. *arXiv preprint arXiv:1704.06199*, 2017.

[24] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[25] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*, 2015.

[26] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[27] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015. URL http://networkrepository.com.

[28] Srijan Kumar, Francesca Spezzano, VS Subrahmanian, and Christos Faloutsos. Edge weight prediction in weighted signed networks. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 221–230. IEEE, 2016.

[29] Srijan Kumar, Bryan Hooi, Disha Makhija, Mohit Kumar, Christos Faloutsos, and VS Subrahmanian. Rev2: Fraudulent user prediction in rating platforms. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 333–341. ACM, 2018.

[30] Tore Opsahl and Pietro Panzarasa. Clustering in weighted networks. *Social Networks*, 31: 155–163, 2009.

[31] Tore Opsahl. Triadic closure in two-mode networks: Redefining the global and local clustering coefficients. *Social Networks*, 35(2):159 – 167, 2013. ISSN 0378-8733.

[32] Ashwin Paranjape, Austin R. Benson, and Jure Leskovec. Motifs in temporal networks. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, WSDM '17, page 601–610, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450346757. doi: 10.1145/3018661.3018731. URL `https://doi.org/10.1145/3018661.3018731`.

[33] Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.

[34] Edoardo M Airoldi, David M Blei, Stephen E Fienberg, and Eric P Xing. Mixed membership stochastic blockmodels. *Journal of machine learning research*, 9(Sep):1981–2014, 2008.

[35] Donatello Conte, Pasquale Foggia, Carlo Sansone, and Mario Vento. Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(03):265–298, 2004.

[36] Giannis Nikolentzos, Giannis Siglidis, and Michalis Vazirgiannis. Graph Kernels: A Survey. *arXiv preprint arXiv:1904.12218*, 2019.

[37] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

A EXTRA EXPERIMENT RESULTS WITH REAL DATASETS

A.1 HISTOGRAM OF SIMILARITIES

See Figure S1.

B EXTRA EXPERIMENT RESULTS WITH SYNTHETIC DATASETS

B.1 GRAPH SIZE COMPARISON

See Figure S2.

B.2 HISTOGRAM OF SIMILARITIES

See Figure S3.

B.3 SOME EXAMPLES OF PREDICTED GRAPHS

See Figure S4, S5, S6, S7, S8, S9, respectively for Path graphs, Ladder graphs with small size, Ladder graphs with large size, Cycle graphs, Path graphs with removal, Cycle graphs with adding extra structures.
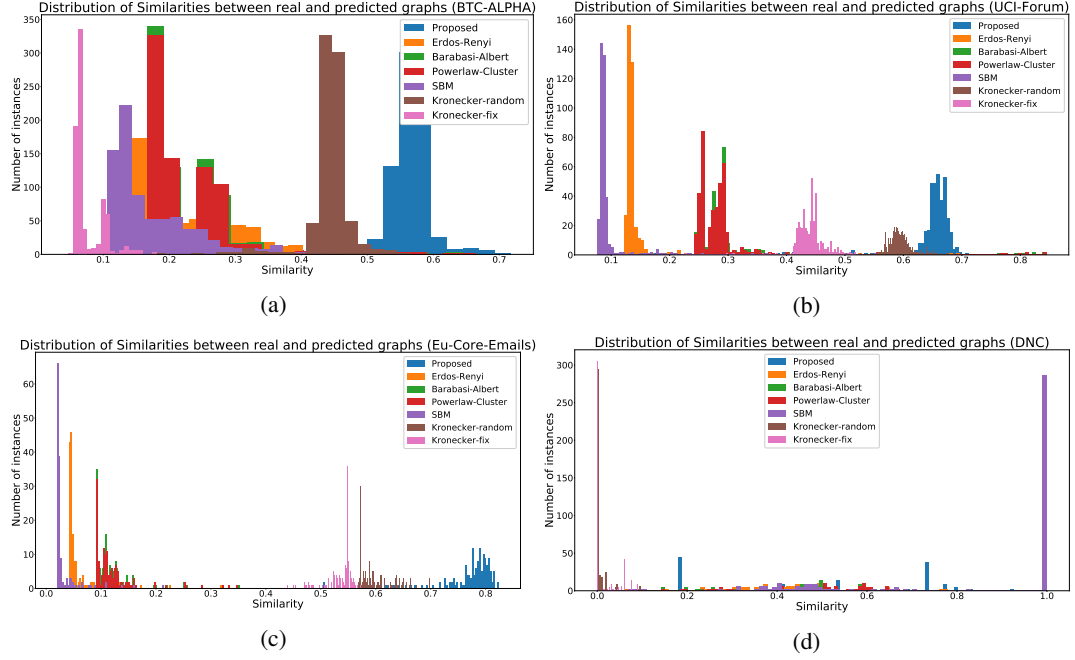
Figure S1: Similarity histograms on real datasets. Blue one is the result of EvoNet, which is compared against 6 random graph models. S1a: BTC-Alpha dataset; S1b: UCI-Forum dataset; S1c: Emails Eu-Core dataset; S1d: DNC emails dataset.
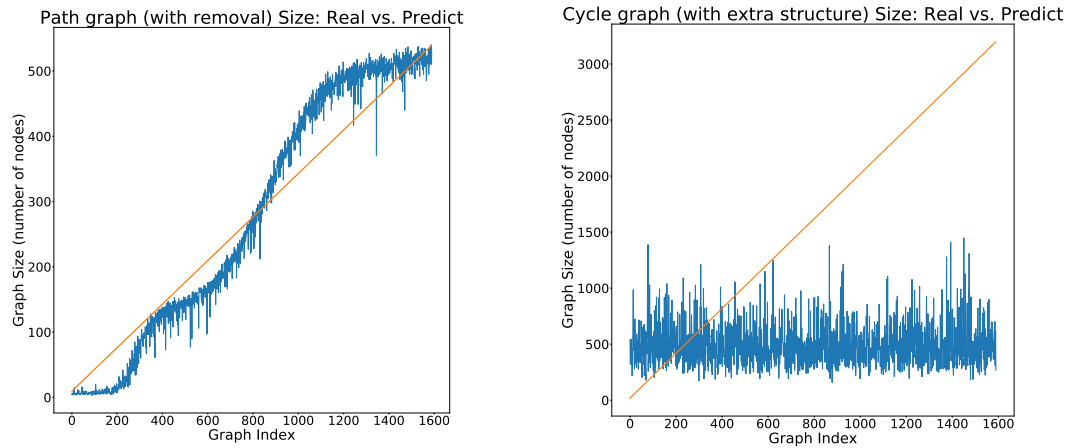
Figure S2: Comparison of graph size: predicted size (blue) VS. real size (orange). Left: Path graphs with removal; Right: Cycle graphs with adding extra structures
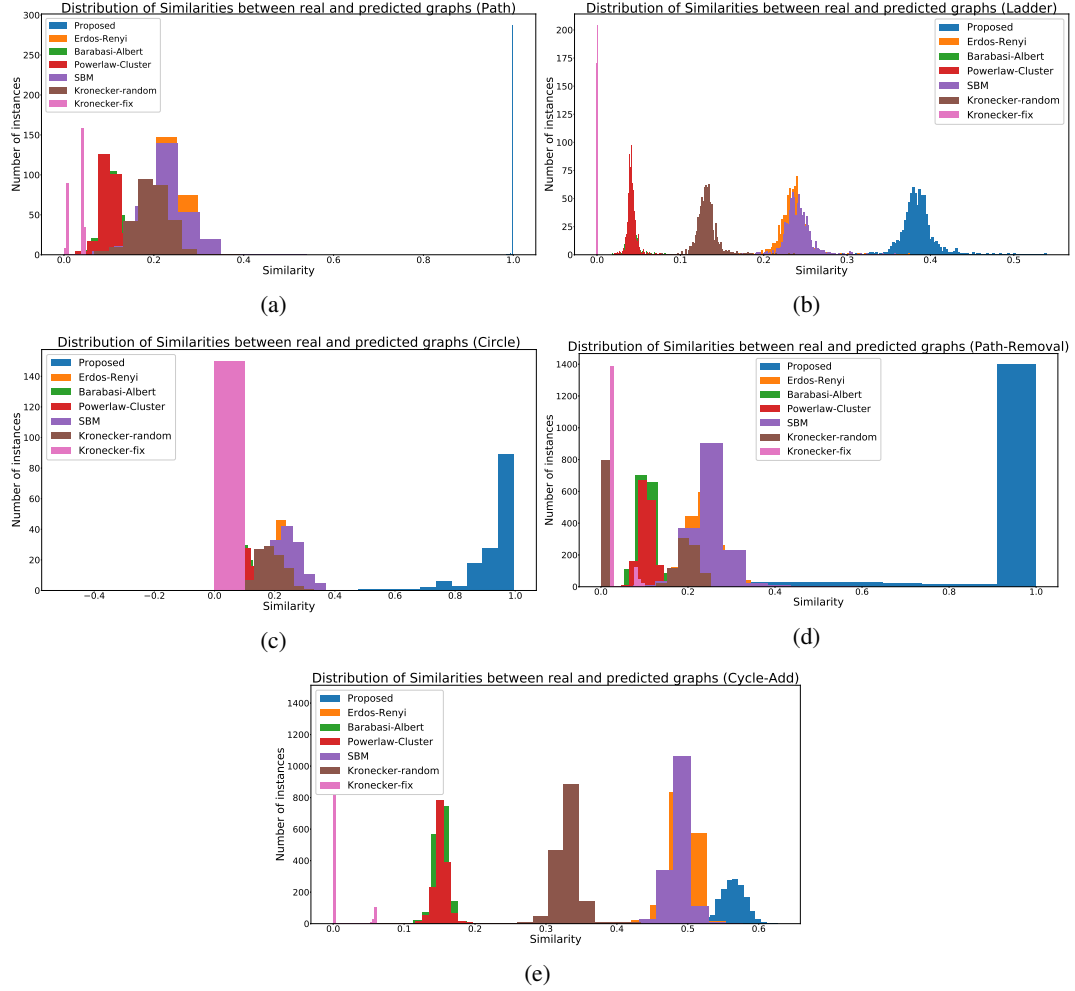
Figure S3: Similarity histograms on synthetic datasets. Blue one is the result of EvoNet, which is compared against 6 random graph models. S3a: Path graphs; S3b: Ladder graphs; S3c: Cycle graphs; S3d: Path graphs with removal; S3e: Cycle graphs with adding extra structures.

Real graphs   Predicted graphs
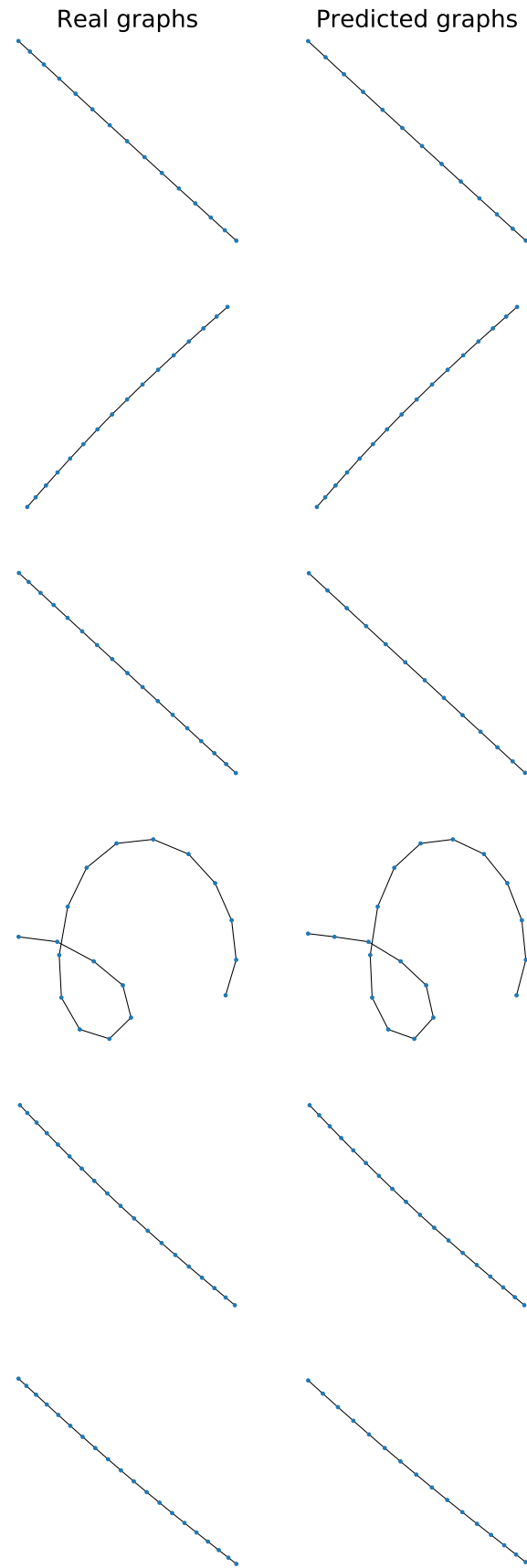
Figure S4: Some examples of predictions on Path datasets: the left column is the real graphs and the right column is the predicted ones.
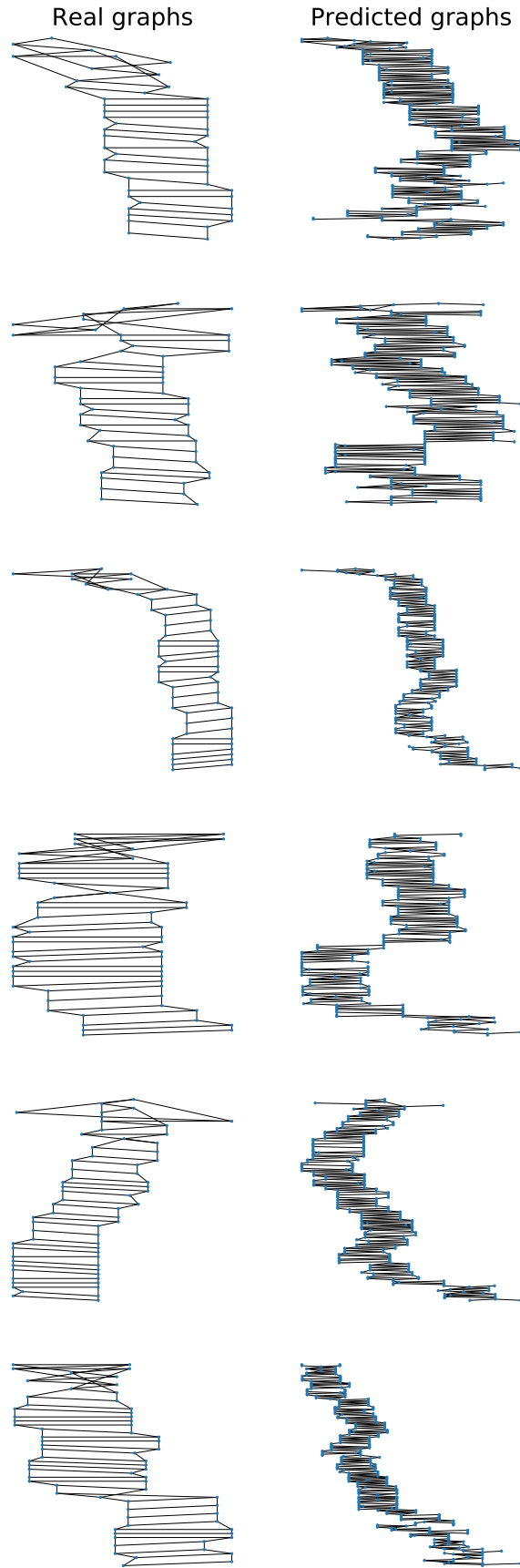
Real graphs   Predicted graphs

Figure S5: Some examples of predictions on Ladder datasets (with small size graphs): the left column is the real graphs and the right column is the predicted ones.
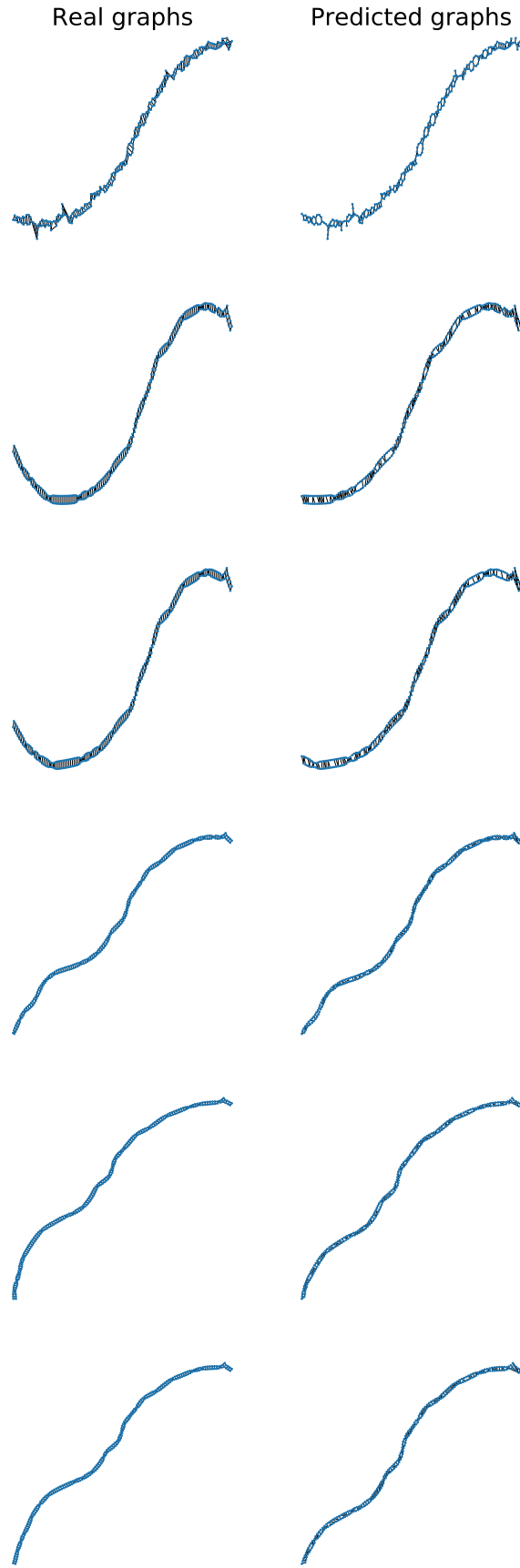
Real graphs   Predicted graphs

Figure S6: Some examples of predictions on Ladder datasets (with large size graphs): the left column is the real graphs and the right column is the predicted ones.
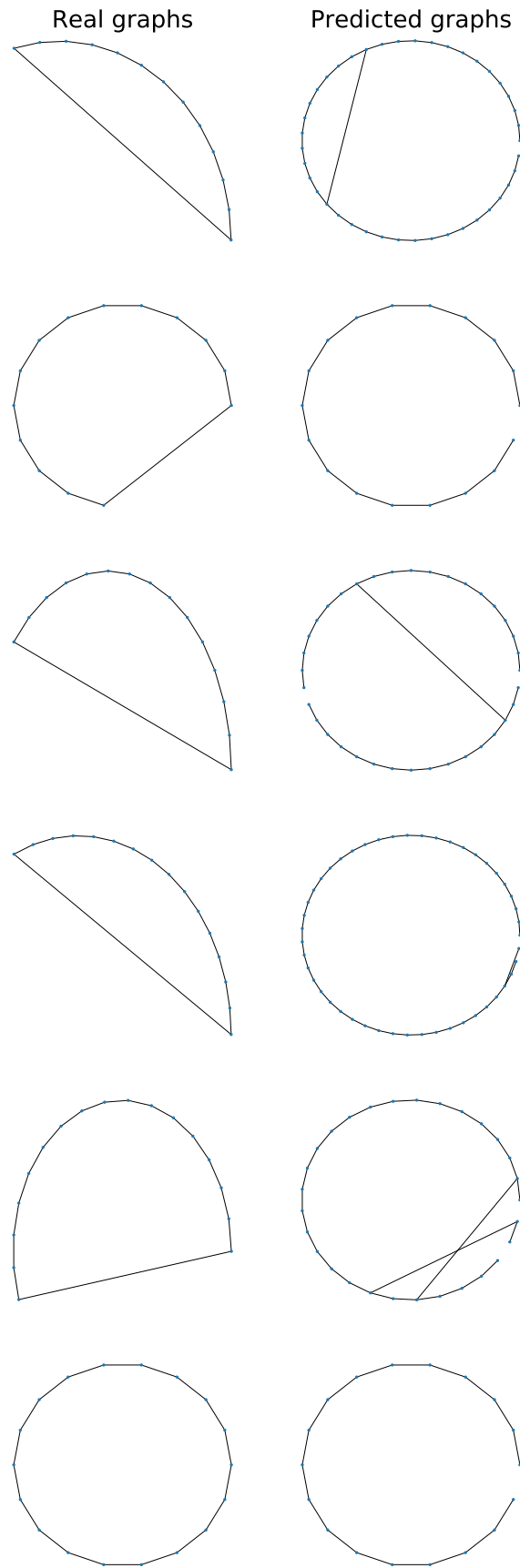
Real graphs   Predicted graphs

Figure S7: Some examples of predictions on Cycle datasets: the left column is the real graphs and the right column is the predicted ones.
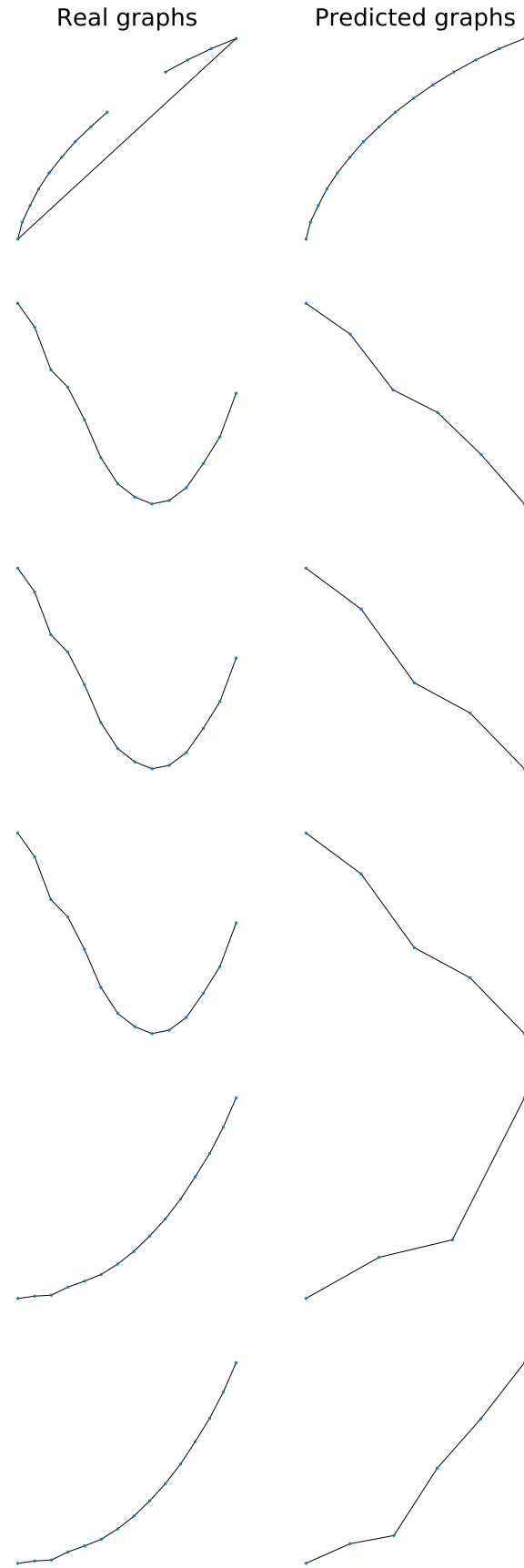
Real graphs   Predicted graphs

Figure S8: Some examples of predictions on Path datasets with removal: the left column is the real graphs and the right column is the predicted ones.
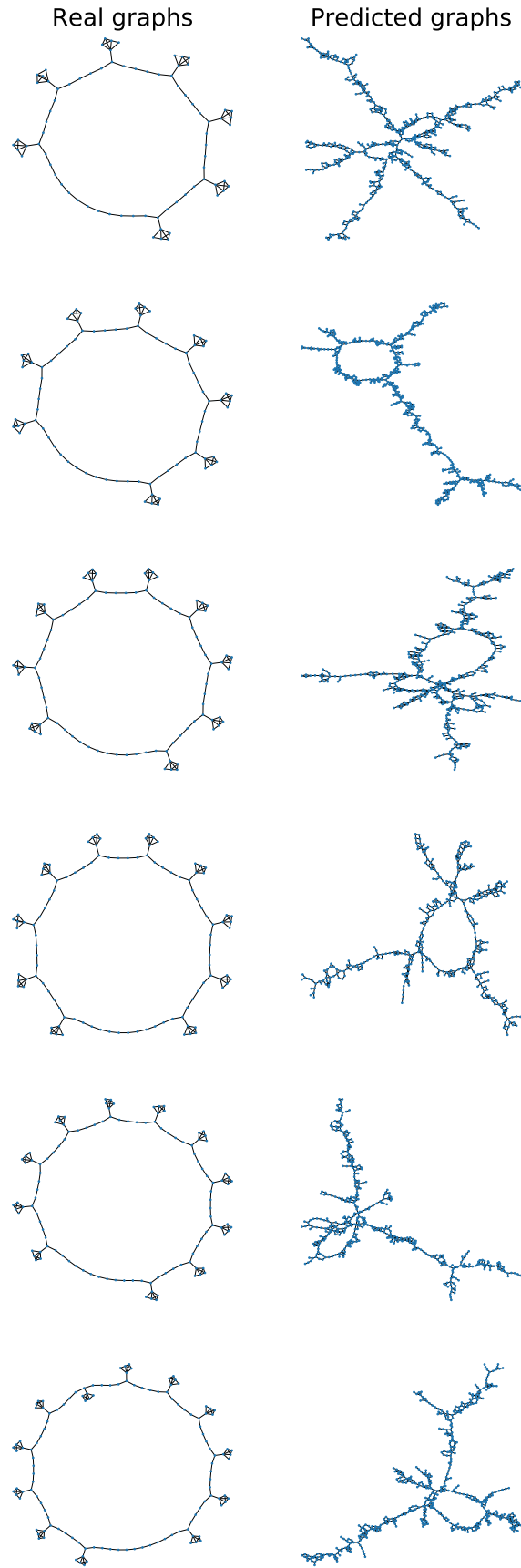
Real graphs　　　　Predicted graphs

Figure S9: Some examples of predictions on Cycle datasets with adding extra structures: the left column is the real graphs and the right column is the predicted ones.