Information Sciences 589 (2022) 250–264

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Hierarchical recurrent neural networks for graph generation

Song Xianduo [a,c,1], Wang Xin [c,d,1], Song Yuyuan [a,c], Zuo Xianglin [b,c,*], Wang Ying [a,b,c,*]

[a] College of Software, Jilin University, Changchun 130012, China
[b] College of Computer Science and Technology, Jilin University, Changchun 130012, China
[c] Key Laboratory of Symbol Computation and Knowledge Engineering (Jilin University), Ministry of Education, Changchun 130012, China
[d] School of Artificial Intelligence, Jilin University, Changchun 130012, China

ARTICLE INFO

Article history:
Received 10 January 2021
Received in revised form 7 December 2021
Accepted 19 December 2021
Available online 27 December 2021

2021 MSC:
00-01
99-00

Keywords:
Graph generation
Edge sequences
Domain-agnostic model
Edge-level RNN
Element-level RNN
Minimum depth-first-search Code

ABSTRACT

Graph generation is widely used in various fields, such as social science, chemistry, and physics. Although the deep graph generative models have achieved considerable success in recent years, some problems still need to be addressed. First, some models learn only the structural information and cannot capture the semantic information. Second, many existing models cannot be applied to large-scale datasets owing to a complex training process and a large number of parameters to be learned. Third, most of the existing models require domain-specific rules to be set up, resulting in poor generalization. To address the aforementioned problems, we propose a domain-agnostic model with hierarchical recurrent neural networks, named GHRNN, which learns the distribution of graph data for generating new graphs. First, we utilize the minimum depth-first-search code to convert a graph into a unique edge sequence that naturally generalizes to various graphs. Then, we adopt the edge-level recurrent neural network (RNN) module to learn the global structure of a graph and the element-level RNN module to obtain semantic information, such as node labels, edge labels, and the timestamp of nodes. The element-level RNN module with a fixed and small number of time steps in GHRNN can efficiently scale to large datasets. Finally, we generate a large number of graphs and evaluate the effectiveness of our model by using 11 different metrics on diverse datasets ranging from molecules to citation networks. Extensive experimental results indicate that GHRNN can capture complex dependencies and generate the most realistic and diverse graphs, in contrast to several state-of-the-art baselines.

© 2021 Elsevier Inc. All rights reserved.

1. Introduction

Graph-structured data can be found in a wide range of applications, such as social networks [40], molecule networks [24,31], energy networks [25], and knowledge graphs [9]. Unlike image and text data, graph-structured data pose unprecedented challenges in machine learning owing to their inherent complexity with non-Euclidean space. In a typical generation task, the provided data distributions are learned to generate real data. Graph generation is more challenging than image and

* Corresponding authors at: College of Software, Jilin University, Changchun 130012, China (W. Ying).
E-mail addresses: songxd19@mails.jlu.edu.cn (S. Xianduo), xinwang@jlu.edu.cn (W. Xin), songyy5517@jlu.edu.cn (S. Yuyuan), zuoxl17@mails.jlu.edu.cn (Z. Xianglin), wangying2010@jlu.edu.cn (W. Ying).
[1] Co-first authors.

https://doi.org/10.1016/j.ins.2021.12.073
0020-0255/© 2021 Elsevier Inc. All rights reserved.

text generation, because it cannot use structured data, such as grids. Graph generation methods [38,19,29] have recently gained considerable attention and merged with new applications such as molecule designing [24,31] and drug discovery [24].

Recent advances in the deep generative models [24,38,19,5] on graph data are essential for improving the fidelity of generated graphs and paving the way for novel applications. The existing methods can be classified into three major categories: generative adversarial network [18] (GAN)-based [29,21,10,37], variational auto-encoder (VAE)-based [24,31,26,30], and recurrent neural network (RNN)-based models [38,19,28]. For the GAN-based models, the GANs are based on a game theory scenario with a min–max game, in which a discriminator and a generator compete against each other. These methods utilize the framework of GANs to capture the distribution of the real dataset and generate data that are highly realistic. CONDGEN [37] generates an entire adjacency matrix of graphs and then converts the adjacency matrix into the graph. To generate molecular graphs, ORGAN [21] and MolGAN [10] apply some chemical rules in molecule generation processes. For the VAE-based models, the VAE [26] pairs a top-down generator with a bottom-up inference network. Training is performed by optimizing the tractable evidence lower bound instead of directly performing maximum likelihood estimation on the intractable marginal log-likelihood. GraphVAE [31] generates graphs with node and edge labels by adding an additional node attribute matrix and an edge attribute matrix. As the molecule graph, GraphVAE, NeVAE [30], and JTVAE [24] also need to establish chemical rules. For the RNN-based models, some models, such as GraphRNN [38], GRAPHGEN [19], and DeepGMG [28], consider graph generation as a sequential generation process. Thus, these model need to convert graphs into sequences before the training process. The simplified molecular-input line-entry system (SMILES) [34] is a standard string-based representation for molecules, and it converts the molecular graphs into the corresponding sequences. In addition, GraphRNN leverages the breadth-first-search (BFS) node orderings to reduce the number of node permutations and generates graphs by adding a sequence of edges and nodes. Although the aforementioned methods have achieved good performance, they still have some inherent limitations and challenges, which are discussed as follows.

Complex Semantic Information. Certain deep graph generative models, such as GraphRNN [38] and NetGAN [5], can learn only the structural distribution of graph data. However, the labels of nodes and edges contain rich semantic information, which is always disregarded in graph generation. In molecular graphs, node labels represent the atomic types, edge label represent a certain chemical bond, and two atoms are more likely to be connected by a bond.

Model Scalability. In a graph with n nodes, the order number of a node can be any digit from 1 to n, including 1 and n. Hence, the graph may have $n!$ different node permutations and various representations, resulting in a large number of training samples and a long time for training. Furthermore, some deep graph generative models require an additional graph matching process after the graph generation process, and this always requires a significant amount of time. For some GAN-based and VAE-based graph generative models, the training time has an exponential relation with the number of nodes or edges. Therefore, these models are not scalable to complex and large graphs. In particular, some GAN-based models often suffer from model collapse due to the adversarial style.

Domain-agnostic Modeling. For some existing molecular graph generative models, guiding the molecular graph generation process by conditioning it on additional chemical rules and SMILES is crucial. These rule-based graph generative models are not appropriate for other types of graphs, such as citation networks and social networks.

To employ deep generative models more effectively and comprehensively for graph generation, we need to answer the following three questions: 1) How to design a model that effectively captures the complex structural and semantic information in graphs? 2) How to reduce the number of node permutations and design a model that performs well in scalability? 3) How to design a model that can generate various graphs without any domain-specific rules?.

To answer these questions, we propose a novel graph generative model based on hierarchical RNNs, named GHRNN, which learns the realistic distribution and performs the task of effectively generating new graphs without any domain-specific rules. Specifically, we study the edge-level and element-level RNN modules to capture complex structural and semantic information of graphs, respectively. For model scalability, we use a fixed and small number of time steps in the element-level RNN module, which only needs a short training time. We utilize the minimum depth-first-search (DFS) code [19] to convert graphs into edge sequences, ensuring that GHRNN does not need any domain-specific rules. Thus, GHRNN can be widely applied to all types of graphs, from molecular graphs to citation networks. Finally, we employ 11 evaluation metrics [33], including structural metrics, label metrics, and similarity metrics, which are used to evaluate the quality of generated graphs in different views. The major contributions of this study are summarized as follows:

- We propose a domain-agnostic graph generation framework named GHRNN with hierarchical RNNs, and it can capture complex structural and semantic information and generate realistic graphs.
- We evaluate the element-level RNN with a fixed and small number of time steps to ensure that the training process of GHRNN is faster than that of other deep graph generative models.
- We propose a new method for evaluating the scalability of models and ensure a fair comparison of models. The experimental results show that GHRNN performs well in terms of most evaluation indicators.

The remainder of this paper is organized as follows. Related studies are described in Section 2. The preliminaries and problem statements of our study are introduced in Section 3. Our proposed model is presented in Section 4. The experimental results are analyzed and discussed in Section 5. Finally, the conclusion and future research directions are provided in Section 6.

2. Related work

In recent years, graph generation has been widely applied in molecular designing [24,31,29]. Graph generative models have been used to generate a large number of molecules and select the molecules having the best chemical property. Graph generation can also be used in computer vision. Researchers extract the objects from the images and generate graphs for images on the basis of the extracted objects. The semantic information in the images can be captured and analyzed more effectively. Most of the graph generative models proposed in recent years have been combined with deep learning, and they have achieved excellent performance in graph generation. Deep learning has been applied in many graph-related studies, such as knowledge-graph embedding [9], community detection [40], and electricity-load forecasting [14,16]. Based on deep generative models, existing graph generative methods can be classified into three categories: GANs-based, VAE-based, and RNN-based.

GANs. These have been successfully applied in many research fields, such as discrete distribution generation [32], information credibility evaluation [35], adversarial attacks [7], image generation [13], and graph generation. NetGAN [5] was the first to successfully generate citation networks by using the architecture of GANs. Specifically, NetGAN adopts a long short-term memory (LSTM) [22] as the generator and adds a new node at every time step of the LSTM. The discriminator of NetGAN is also based on the architecture of the LSTM. At every time step, a one-hot vector of the node at the current position is fed as input. After all nodes in a graph are processed, the discriminator outputs a score that denotes the probability of the graph being real. NetGAN pays attention to the structure of graphs and learns the structural distribution of graphs. However, most of the graphs have node labels and edge labels. For example, molecules can be represented as graphs with node labels and edge labels. Unlike NetGAN, ORGAN [21] uses the SeqGAN [39] framework to optimize the chemical properties of molecules by using a linear combination of a discriminator. MolGAN [10] selects a multilayer perceptron (MLP) to generate an entire molecular graph instead of selecting RNNs as a generator to generate graphs iteratively. CONDGEN [37] uses a graph convolution network to obtain graph embeddings and generates graphs with certain structures by utilizing a condition vector. Mol-CycleGAN [29], a generative model based on CycleGAN [41] for molecular optimization, can be trained to learn the transformation from one style to another.

VAEs. The models based on VAEs, such as JTVAE [24], GraphVAE [31], and NeVAE [30], use an encoder to encode molecular graphs. Then, the latent vectors are decoded into molecules by using a decoder. By optimizing the objective function composed of reconstruction errors and KL-divergence, the decoder can generate target vectors based on implicit vectors sampled from a multi-dimensional standard Gaussian distribution. GraphVAE first generates a probabilistic fully connected graph and then uses a standard graph-matching algorithm to match the ground truth. JTVAE produces a tree-structured scaffold over chemical substructures and then incorporates the substructures into a complete molecule. However, the standard graph-matching algorithm requires a substantial amount of time. Some VAE-based graph generative models require the algorithm to be enforced, rendering them non-scalable. Additionally, some VAE-based models can generate only small graphs such as molecular graphs.

RNNs. These were specially designed for processing sequence data and have made considerable progress in graph generation. For example, GraphRNN [38], DeepGMG [28], and molecule generation [4] first convert graphs into sequences and utilize RNNs to generate graph sequences. Then, the generated sequences can be mapped to the corresponding graphs. For better scalability, GraphRNN converts graphs into graph sequences by performing BFS traversal and introduces a BFS node ordering scheme, which reduces the number of possible node permutations and the time complexity of training. However, a graph may still have many possible node permutations and GraphRNN consumes a substantial amount of time on large datasets. GRAPHGEN [19] maps a graph to a unique sequence to reduce the complexity of training. It achieves better scalability than GraphRNN and DeepGMG. However, graphs contain rich semantic information and global structural information. Some RNN-based graph generative models cannot capture this information and, thus, cannot learn the real distribution of graphs.

3. Preliminary and problem definition

In this section, we provide a brief overview of the minimum DFS code and formalize the problem of deep graph generation of GHRNN. The main notations are summarized in Table 1, including those to be introduced in Section 4.

3.1. Minimum DFS code: Graphs to edge sequences

The DFS code [36] can convert graphs into edge sequences by performing a DFS traversal, which starts at any node in the graph. During the DFS traversal process, when a node is traversed, it is assigned a timestamp. According to the timestamp assigned to the node, edge (u, v) can be represented by a 5-tuple. The 5-tuples of these edges form a part of the sequence representation of the graph. For example, the DFS traversal order depicted in Fig. 1b is $(v_0, v_1), (v_1, v_2), (v_2, v_3), (v_2, v_4)$. If edge (u, v) is a part of the DFS traversal path, it is called a forward edge, which must satisfy $t_u < t_v$, i.e., node u is always traversed before node v. By contrast, the dashed lines in Figs. 1b and 1c are called backward edges and are not a part of DFS traversal. The timestamps of edges (u, v) always indicate $t_u > t_v$. Although these backward edges do not belong to the DFS traversal paths, the complete sequence representation of the graph should include the 5-tuples of these backward edges.

Table 1
Notations.

Describe	Formal representation	Annotation
Node	v	
Graph	G	
Generated graph	$\hat{G}$	
Node label set	$\mathscr{V}_L$	
Edge label set	$\mathscr{E}_L$	
Probability distribution	$p(\cdot)$	
Node set	$\mathscr{V} = \{v_1, \ldots, v_n\}$	n is the number of nodes.
Edge set	$\mathscr{E} = \{(v_i, v_j) \mid v_i, v_j \in \mathscr{V}\}$	
Edge sequence	$S = \{s_1, \ldots, s_m\}$	m is the number of 5-tuples in the edge sequence.
Generated edge sequence	$\hat{S} = \{\hat{s}_1, \ldots, \hat{s}_m\}$	
Graph set	$\mathscr{G} = \{G_1, G_2, \ldots, G_N\}$	N is the number of graphs in the set.
Generated graph set	$\hat{\mathscr{G}} = \{\hat{G}_1, \hat{G}_2, \ldots, \hat{G}_N\}$	
Edge sequence set	$\mathscr{S} = \{S_1, S_2, \ldots, S_N\}$	
Generated edge sequence set	$\hat{\mathscr{S}} = \{\hat{S}_1, \hat{S}_2, \ldots, \hat{S}_N\}$	
Start token	SOS	Start sign of the graph generation.
End token	EOS	End sign of the graph generation.
Transformation function	$\mathscr{F}(G) \to S$	Denotes the process of converting graphs into sequences.
Inverse transformation function	$\mathscr{F}^{-1}(\hat{S}) \to \hat{G}$	Denotes the process of converting sequences into graphs.
Categorical distribution	$Cat(\theta)$	θ is the parameter of the distribution.
Edge-level RNN module	$f_{trans0}(\cdot)$	
Element-level RNN module	$f_{trans1}(\cdot)$	
5-tuple sequence representation of an edge	$s = (t_p, t_q, L_p, L_e, L_q)$	t_p, t_q, L_p, L_e, and L_q denote the timestamp of node p, timestamp of node q, label of node p, label of edge e, and label of node q, respectively.

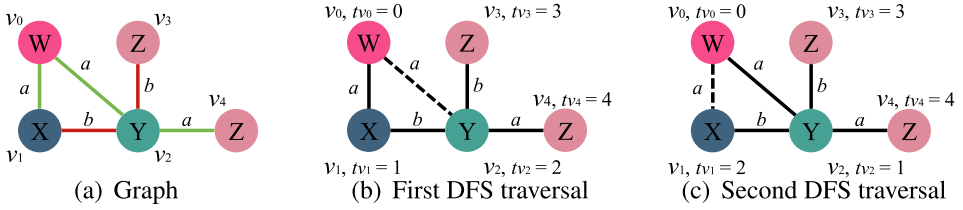


Fig. 1. Generation process of the DFS code. In sub-figure (a), the capital letter in the node denotes the node label, and the green and red edges represent edges with labels a and b, respectively. Sub-figures (b) and (c) are two different DFS traversal modes starting from node v_0. The blond lines denote forward edges, which are the trajectory of DFS traversal, and the dashed lines denote backward edges, which are not a part of DFS traversal.

To incorporate backward edges into the complete sequence, we follow three rules: 1) backward edge (u, u') must be ahead of all forward edges with u as the head node; 2) all forward edges with u as the tail node should be arranged before the backward edge with u as the head node; 3) in backward edges (u, u') and (u, v') with u as the head node, (u, u') should precede (u, v') if $t_{u'} < t_{v'}$. Based on the aforementioned rules, we can obtain the complete sequence representation of the graph. This sequence representation is referred to as the DFS code.

To explain the DFS code in detail, the graph shown in Fig. 1(a) is used as an example. In Fig. 1(b), the DFS code of the graph is $[(0,1,W,a,X)\,(1,2,X,b,Y)\,(2,3,Y,b,Z)\,(2,4,Y,a,Z)\,(2,0,Y,a,W)]$. In Fig. 1(c), the DFS code of the graph is $[(0,1,W,a,Y)\,(1,2,Y,b,X)\,(2,0,X,a,W)\,(1,3,Y,b,Z)\,(1,4,Y,a,Z)]$. Fig. 1(b) and (c) show only two different DFS codes obtained by two different DFS traversals; the graph shown in Fig. 1(a) may have more different DFS codes. When there are many different DFS codes, the graph generation process becomes longer and more complex. Therefore, we explore the lexicographical ordering [36] to find the smallest DFS code from many DFS codes for the graph. The DFS code with the smallest lexicographical ordering is the minimum DFS code. For example, the DFS code for Fig. 1(b) is smaller than that for Fig. 1(c) because the first 5-tuple $(0,1,W,a,X)$ of Fig. 1(b) is smaller than that $(0,1,W,a,Y)$ of Fig. 1(c) as $X < Y$. We can provide a unique sequence representation for a graph by using the minimum DFS code, which makes the graph generation process simpler and more efficient. Here, the edge sequence of a graph refers to its minimum DFS code.

3.2. Problem definition

A set of observed graphs, $\mathscr{G} = \{G_1, G_2, \cdots, G_M\}$, is sampled from a graph data distribution, $p(\mathscr{G})$, where each graph G_i consists of a node set, $\mathscr{V} = \{v_1, \ldots, v_n\}$, and an edge set, $\mathscr{E} = \{(v_i, v_j) \mid v_i, v_j \in \mathscr{V}\}$. In graph set $\mathscr{G}$, $\mathscr{V}_L$ and $\mathscr{E}_L$ denote the set of node labels and edge labels, respectively. $\mathscr{S} = \{S_1, S_2, \cdots, S_M\}$ is defined as an edge sequence set, where S_i is the edge sequence of graph G_i. Each edge in edge sequence S_i is represented by a 5-tuple, $(t_p, t_q, L_p, L_{(p,q)}, L_q)$, where $t_p, t_q, L_p, L_{(p,q)}$,

and L_q denote the timestamp of node p, timestamp of node q, label of node p, label of edge (p, q), and label of node q, respectively. For simplicity, $L_{(p,q)}$ is replaced by L_e in the remainder of this paper.

The aim of deep graph generation is to learn the distribution of the graphs and generate realistic graphs. Fig. 2 shows the generation process of GHRNN. First, we use the minimum DFS code and a transformation function, $\mathscr{F}(\cdot)$, that converts graphs into unique sequence representations, $\mathscr{F}(\mathscr{G}) \rightarrow \mathscr{S}$. Then, the edge sequences are fed into GHRNN for learning the distribution of graphs indirectly. After the learning process is completed, GHRNN can generate edge sequences, $\hat{\mathscr{S}}$. Each graph corresponds to a unique edge sequence, and each edge sequence corresponds to a unique graph. Therefore, the aforementioned function is invertible, i.e., $\mathscr{F}^{-1}(\hat{\mathscr{S}}) \rightarrow \hat{\mathscr{G}}$. GHRNN generates an edge sequence and naturally converts it into a unique graph.

4. Proposed method

In this section, first, we introduce our proposed framework, GHRNN, and consider molecule generation as an example to describe the generation process of graphs. Then, we present each component in detail, namely, edge-level RNN module and element-level RNN module. Finally, we describe the training process of the model.

4.1. GHRNN framework

4.1.1. Architecture of GHRNN

Because various types of graphs can be converted into edge sequences by using the minimum DFS code described in Section 3.1, we can transform the learning of the distribution of graphs to learning of the distribution of edge sequences. The architecture of GHRNN is illustrated in Fig. 3; it learns the distribution of graph data from edge sequences of graphs and generates the edge sequences of new graphs. GHRNN with a hierarchical RNN structure can effectively capture the structural and semantic information of graphs. The initial input of GHRNN is *SOS*. The new hidden states generated by the edge-level RNN module contain new structural information about graphs (i.e., the edges in the dotted line). The hidden state of element-level RNN module is initialized by the edge-level RNN module. The element-level RNN module captures the semantic information of node and edge labels, generating the elements of labels and timestamps to form a complete edge. Each generated edge is converted into the input of the next time step of the edge-level RNN by applying the embedding function, f_{emb_s}. Every element has one additional dimension to incorporate the end token *EOS*. If the hidden state of the edge-level RNN does not contain the information of the new edge, the element-level RNN will predict *EOS*, indicating the end of the graph generation process. The generated edges finally form a complete graph.

4.1.2. Molecule generation

In this section, we consider molecule generation as an example to describe the graph generation process of GHRNN in detail. Because the generation process of a molecule is complicated, we illustrate the generation process of only a sample molecule: a water molecule, H_2O.

First, the initial hidden state, h_0, of the edge-level RNN module is initialized by using the method described in [17]. Second, we input the start token *SOS* into the edge-level RNN module at the beginning of the generation process to inform that GHRNN starts the molecule generation process. In our implementation, the start token is a zero vector and the 5-tuple, s_0, is the start token *SOS*. To limit the figure size, we do not show the start token *SOS* and hidden state h_0 in Fig. 4.

Subsequently, the edge-level RNN module generates hidden state h_1 conditioned on the initial hidden state, h_0, and the start token *SOS*. The element-level RNN module can generate the first element, t_p, conditioned on hidden state h_1. However, we do not consider the raw output, θ_{t_p}, of the element-level RNN module as the final result. Node timestamp t_p is randomly sampled from the categorical distribution parameterized by θ_{t_p}. For other elements, we implement this same strategy. As shown in the Fig. 4, the first generated element is a node timestamp, t_p, i.e., node 0. Next, the initial hidden state $h_{1,0}$ of the element-level RNN module is initialized with the hidden state h_1 of the edge-level RNN module. The element-level RNN module obtains the new hidden state, $h_{1,1}$, conditioned on the hidden state, $h_{1,0}$, and the generated element, t_p. It generated a new element, t_q, according to the new hidden state, $h_{1,1}$. For this example, the second generated element is also a node timestamp, t_q, i.e., node 1. The element-level RNN module recurrently executes the aforementioned process until a complete edge is generated.

The lowercase and uppercase alphabets in the rounded rectangle represent the type of the generated chemical bonds and atoms, respectively. For instance, the lowercase "s" denotes a single bond and the uppercase "H" denotes a hydrogen atom.

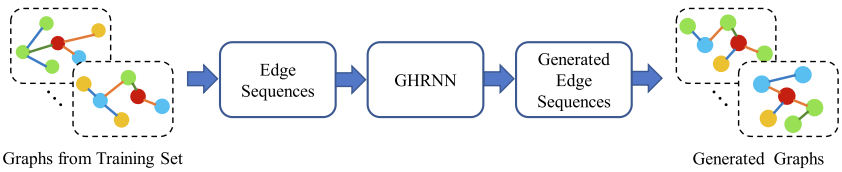


Fig. 2. Generation Process of GHRNN.

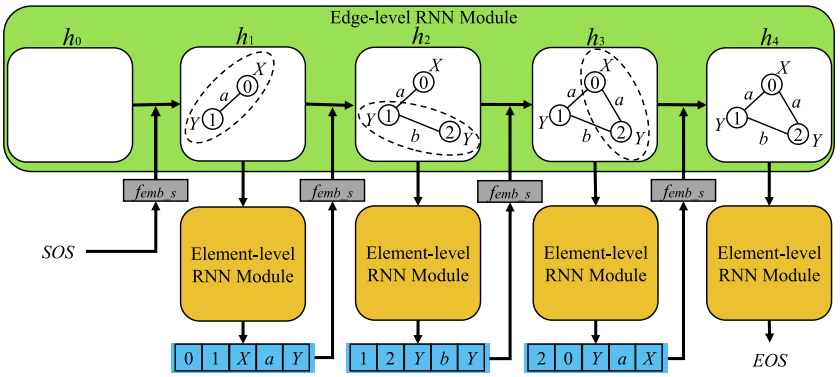


Fig. 3. Architecture of GHRNN. The two main components of our GHRNN can be summarized as follows: (1) edge-level RNN module learns the global structure of graphs, generating the new hidden state conditioned on the new input and the current hidden state; (2) element-level RNN module captures the semantic information of node and edge labels, generating the elements of labels and timestamps to form a complete edge.

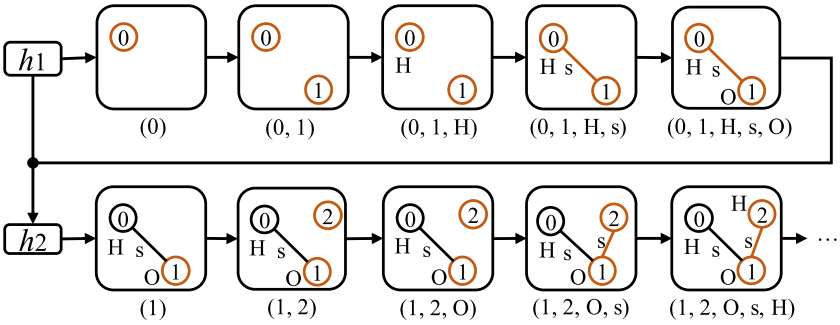


Fig. 4. Depiction of the generation process of water molecule.

We numerically represent the categories of chemical bonds and atoms in our code. Here, the reason that the types of atoms and chemical bonds are represented by alphabets instead of numbers is that we want to distinguish chemical bonds from atoms. This can help readers better understand the molecule generation process. When other elements are generated, all generated elements form a complete edge. The edge-level RNN module can obtain the new hidden state, h_2, according to the hidden state, h_1, and the generated edge, $(0, 1, H, s, O)$. Finally, a whole water molecule can be gradually completed by executing the aforementioned operations recurrently until the end tokens *EOS* appear in the generation process.

The end tokens, *EOS*, are several different numbers, which are the indexes of the last dimension of the five elements. For water molecule, the end tokens of t_p, t_q, L_p, L_e, and L_q are $3, 3, 2, 1$, and 2, respectively. This is because a water molecule has three nodes, two types of node labels, and one type of edge label. When t_p generated by GHRNN is 3, the molecule generation process is finished. Note that GHRNN can generate the end tokens of other elements.

4.2. Edge-level RNN module

The edges in graphs not only reflect the structural characteristics, but also express the relationship between nodes. The complex dependencies between nodes and edges pose considerable challenges in graph generation. To simplify the graph generation process, we convert the graphs into the edge sequences and transform the complex dependencies into the dependencies of the 5-tuples, s, in the edge sequence, S. Therefore, how to capture dependencies in the edge sequences and learn the global structural features of the graph is an important problem in graph generation.

The LSTM can effectively solve the problem of long-term dependency between sequences. Therefore, we explore the LSTM to construct the edge-level RNN module, which can capture the structural dependencies between edges. Specifically, we decompose the distribution of the edge sequences as the product of conditional distributions on the edges:

$$p(S) = p(s_1, s_2, \cdots, s_n) = \prod_{i=1}^{n} p(s_i | s_{<i}), \tag{1}$$

where s_i is the 5-tuple representation of the i-th edge, $s_{<i}$ denotes the first $i-1$, which generated 5-tuples, and n is the number of 5-tuples. GHRNN captures the dependencies between the 5-tuples and generates new 5-tuples based on the previously generated 5-tuples. In other words, GHRNN can learn the dependencies between the edges and the global structure of graphs.

The new hidden state, h_i, contains the information of generated subgraphs and new edges. To generate new edges, we need to obtain the new hidden state, h_i, by using the edge-level RNN module:

$$h_i = f_{trans0}(h_{i-1}, f_{emb_s}(s_{i-1})), \tag{2}$$

where f_{emb_s} embeds s_{i-1} into an input vector for the edge-level RNN module, f_{trans0}. The dimension of the input vector should be the same as that of the edge-level RNN's input. The edge-level RNN module, f_{trans0}, can predict the new hidden state, h_i, according to the hidden state vector, h_{i-1}, and the input vector. Hence, the edge-level RNN module can capture the edge dependencies and the global structure of graphs. In the following section, we discuss how to use the element-level RNN module to learn element dependencies in detail.

4.3. Element-level RNN module

The gated recurrent unit (GRU) also performs well in processing sequence data. Moreover, it has fewer parameters and faster convergence than the LSTM in the training process. Therefore, we utilize the structure of the GRU to develop the element-level RNN module. We apply it to capture the element dependencies. Fig. 5 shows the unfolded element-level RNN module. As each tuple includes five elements, we further decompose the conditional distribution $p(s_i|s_{<i})$ expressed in Eq. 1 as the product of conditional distributions over elements:

$$\begin{aligned} p(s_i|s_{<i}) &= p\big((t_p, t_q, L_p, L_e, L_q)|s_{<i}\big) \\ &= p\big(t_p|s_{<i}\big) \times p\big(t_q|s_{<i}, t_p\big) \times p\big(L_p|s_{<i}, (t_p, t_q)\big) \\ &\quad \times p\big(L_e|s_{<i}, (t_p, t_q, L_p)\big) \times p\big(L_q|s_{<i}, (t_p, t_q, L_p, L_e)\big). \end{aligned} \tag{3}$$

Unlike the edge-level RNN module, the element-level RNN module involves only a small and fixed number of time steps. Particularly, the hidden state of the edge-level RNN module is updated once, whereas the hidden state of the element-level RNN module is updated four times. We define the process as the following functions:

$$\begin{aligned} h_{i,0} &= f_{trans1}\Big(h_i, f_{emb_t_p}(t_p)\Big), \qquad h_{i,1} = f_{trans1}\Big(h_{i,0}, f_{emb_t_q}(t_q)\Big), \\ h_{i,2} &= f_{trans1}\Big(h_{i,1}, f_{emb_L_p}(L_p)\Big), \quad h_{i,3} = f_{trans1}\big(h_{i,2}, f_{emb_L_e}(L_e)\big), \end{aligned} \tag{4}$$

where $f_{emb_t_p}, f_{emb_t_q}, f_{emb_L_p}$, and $f_{emb_L_e}$ are embedding functions, and f_{trans1} denotes the element-level RNN module. The embedding functions are used to encode the generated elements into the input of the element-level RNN module. The element-level RNN module generates the next element based on the previously generated elements. As a result, the element-level RNN module can efficiently capture the element dependencies.

In this study, we need to generate five elements and then concatenate them at each time step of the edge-level RNN module. These elements can be classified into three categories: node timestamps (t_p, t_q), node labels (L_p, L_q), and an edge label (L_e). Thus, the element-level RNN module should output three different dimensional vectors. However, it is impossible for a single RNN to output three different dimensional vectors. To address this problem, we adopt four output functions to extract all elements except the first element from different hidden states of the element-level RNN. The first element, t_p, is directly predicted based on, the hidden state of the edge-level RNN module and the output function f_{t_p}.

In Section 4.1.2, we mentioned that the elements are sampled from the distribution. If we want to obtain a definite and meaningful distribution, we need to specify parameters for the distribution. The parameters are obtained through the softmax layer that maps multiple scalars to a probability distribution. Furthermore, each value of the softmax layer's output ranges from 0 to 1. We use the output values of the softmax layer to parameterize a probability distribution and sample elements from it. Specifically, the results of the output functions are passed through the softmax layer to generate the output vectors, $\theta_{t_p}, \theta_{t_q}, \theta_{L_p}, \theta_{L_e}$, and θ_{L_q}. $\theta_{t_p}[i]$ and $\theta_{L_p}[j]$ can be interpreted as a probability of the ith timestamp of node p and a probability of label j of node p, respectively. The meaning of $\theta_{t_q}, \theta_{L_e}$, and θ_{L_q} is similar to that of $\theta_{t_p}[i]$ and $\theta_{L_p}[j]$. We sample the elements in the 5-tuple from the categorical distributions parameterized by $\theta_{t_p}, \theta_{t_q}, \theta_{L_p}, \theta_{L_e}$, and θ_{L_q}. Therefore, the element generation process can be represented by the follow equations:

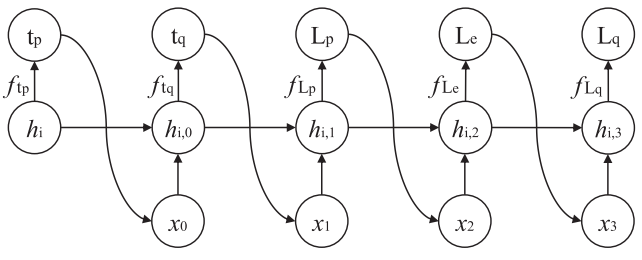


Fig. 5. Architecture of the element-level RNN module.

$$\theta_{t_p} = SOFTMAX\Big(f_{t_p}(h_i)\Big), \quad t_p \sim Cat(\theta_{t_p}), \tag{5}$$

$$\theta_{t_q} = SOFTMAX\Big(f_{t_q}(h_{i,0})\Big), \quad t_q \sim Cat(\theta_{t_q}), \tag{6}$$

$$\theta_{L_p} = SOFTMAX\Big(f_{L_p}(h_{i,1})\Big), \quad L_p \sim Cat(\theta_{L_p}), \tag{7}$$

$$\theta_{L_e} = SOFTMAX(f_{L_e}(h_{i,2})), \quad L_e \sim Cat(\theta_{L_e}), \tag{8}$$

$$\theta_{L_q} = SOFTMAX\Big(f_{L_q}(h_{i,3})\Big), \quad L_q \sim Cat(\theta_{L_q}), \tag{9}$$

where f_{t_p},f_{t_q},f_{L_p},f_{L_e}, and f_{L_q} are the output functions and the elements are sampled from the categorical distributions. We obtain the generated elements by randomly sampling from the categorical distributions. This operation allows the generated graphs to be diverse. The five generated elements are finally concatenated into a whole 5-tuple, s, which can be represented by the following equation:

$$s_i = concat(t_p, t_q, L_p, L_e, L_q). \tag{10}$$

4.4. Model training

Supervised learning is an effective method of machine learning; thus, we use supervised learning in graph generation and define the objective function (i.e., the loss function) in this subsection. For the objective function, we consider the difference between the generated edges and the ground truth. Therefore, we choose the binary cross-entropy loss function as the objective function:

$$BCE(\hat{s}_i, s_i) = -\sum_{c}\Big(s_i[c]^T \log \hat{s}_i[c] + (1 - s_i[c])^T \log(1 - \hat{s}_i[c])\Big), \tag{11}$$

where c is the element in the generated 5-tuple, $\hat{s}_i$, or the ground truth 5-tuple, s_i.

In our code, the five elements are represented in one-hot encoding. Here, we discuss how to determine the dimension of vectors that represent the elements. Given a graph set, $\mathscr{G}$, we find the maximum node set: $\mathscr{V}_{max} = \max_{\forall G(\mathscr{V},\mathscr{E})\in\mathscr{G}} \mathscr{V}$. We set the dimension of the timestamps of nodes to be $|\mathscr{V}_{max}| + 1$. The dimensions of node and edge labels are $|\mathscr{V}_L| + 1$ and $|\mathscr{E}_L| + 1$, respectively. Here, every element has one additional dimension to incorporate one end token. When the index of the last dimension of any of the five elements is predicted, the whole generation process of graphs is completed.

The graph generation strategy of GHRNN is summarized in Algorithm 1. Line 1 initializes edge sequence $\widehat{S}$ of a graph to an empty set. Line 2 initializes hidden state h_0 of the edge-level RNN randomly and assigns *SOS* to the initial 5-tuple, $\hat{s_0}$. Lines 3–19 describe the generation of an edge sequence until the end token *EOS* is generated. Lines 4–6 indicate that the new hidden state of the edge-level RNN is obtained and the hidden state of the element-level RNN with h_i is initialized. Lines 7–17 show that five elements are generated by using different output functions and the elements are concatenated to form an entire 5-tuple, $\hat{s}_i$. In Line 18, the new 5-tuple is incorporated into $\widehat{S}$. Finally, in line 20, the generated edge sequence, $\widehat{S}$, is converted into a graph.

Algorithm 1. Graph Generation Algorithm

Input: Edge-level RNN f_{trans0}, element-level RNN f_{trans1}, five outputs functions f_{t_u}, f_{t_v}, f_{L_u}, f_{L_e}, and f_{L_v}, four embedding functions f_{emb_s}, $f_{emb_t_u}$, $f_{emb_t_v}$, $f_{emb_L_u}$, and $f_{emb_L_e}$, start token *SOS*, and end token *EOS*.
Output: Graph G

1 $\hat{S} \leftarrow \emptyset, i \leftarrow 0$;
2 Initialize $h_i, \hat{s}_i \leftarrow SOS$;
3 **repeat**
4 $i \leftarrow i + 1, \hat{s}_i \leftarrow \emptyset$;
5 $h_i \leftarrow f_{trans0}(h_{i-1}, \hat{s}_{i-1})$;
6 $j \leftarrow -1, h_{i,j} \leftarrow h_i$;
7 **for** $k \in \{t_p, t_q, L_p, L_e, L_q\}$ **do**
8 $\theta_k \leftarrow f_k(h_{i,j})$;
9 $k_j \sim Cat(\theta_k)$;
10 **if** $j == 4$ **then**
11 break ;
12 **else**
13 $j \leftarrow j + 1$;
14 $h_{i,j} \leftarrow f_{trans1}(h_{i,j-1}, f_{emb_k}(\theta_k))$;
15 $\hat{s_i} \leftarrow \text{concat}(\hat{s_i}, k_j)$;
16 **end**
17 **end**
18 $\hat{S} \leftarrow \hat{S} || \langle \hat{s_i} \rangle$;
19 **until** $\exists k_j, k_j = EOS$;
20 **return** $\mathscr{F}^{-1}(\hat{S})$;

5. Experiments

To verify the effectiveness of the proposed framework, GHRNN, we attempt to answer the following questions:

- How effective is the proposed framework, GHRNN, compared with the state-of-the-art domain-agnostic methods on learning the graph distributions of various benchmark datasets?
- How scalable is GHRNN in the graph generation task, compared with the state-of-the-art domain-agnostic methods?

5.1. Datasets

We evaluate our framework by using six widely used datasets, including molecular graphs, protein interaction networks, and citation networks. Datasets Lung and Yeast [1] are molecule datasets obtained from the National Center for Biotechnology Information of the United States National Library of Medicine. The All dataset [19] is composed of different molecules from four other molecule datasets [1]: NCI-H23, MOLT-4, MCF-7, and Yeast. Because the Enzymes [6] has only a few node labels, we can obtain more node labels by combining the node labels with their degree. For citation networks, we attempt to evaluate the deep graph generative model's performance on large graphs. However, both Citeseer and Cora have only a whole large graph, which is not appropriate for graph generation. Because a graph cannot express a probability distribution, we sample a certain amount of subgraphs from Citeseer and Cora via random walks with restarts (RWR). Citeseer-s and Cora-s are all synthetic datasets and sampled from Citeseer and Cora via RWR, respectively. The sampled subgraphs are much larger than the molecular graphs. As a result, we can evaluate the models' performance on large graphs. The statistics of each dataset are summarized in Table 2. $|\mathscr{V}|_{(min,max)}$ and $|\mathscr{E}|_{(min,max)}$ denote the minimum and maximum number of nodes and edges, respectively.

5.2. Baselines

To verify the effectiveness of GHRNN on graph generation, we compare it with two other state-of-the-art baselines: GraphRNN [38] and GRAPHGEN [19]. These are domain-agnostic models and can generate node and edge labels. Some traditional methods, such as Erdős-Rényi [12], Barabási-Albert [3], mixed membership stochastic block models [2], and Kronecker graph models [27], have been outperformed by many deep graph generative models. Therefore, we do not compare GHRNN with those in our experiment.

5.3. Evaluation metrics

For evaluating the generated graphs, the key is to calculate the *maximum mean discrepancy* (MMD) [20] between the ground truth dataset and the generated graph set. Because it is difficult to use only one metric to evaluate the quality of generated graphs, we attempt to use various evaluation metrics. Therefore, we evaluate the quality of generated graphs in terms of three main perspectives: structural difference, distribution discrepancy of labels, and novelty. We utilize five evaluation metrics to measure the structural differences: 1) the distribution of node degrees, 2) clustering coefficient distribution of nodes, 3) orbit count distribution, 4) average number of nodes, and 5) average number of edges. Furthermore, we adopt three metrics to evaluate the distribution discrepancy of labels: 6) distribution of node labels, 7) distribution of edge labels, and 8) joint distribution of node labels and degree. The last metric involves not only the structure of graphs, but also the node labels.

We believe that the generated graphs are novel. Specifically, the generated graphs should be different from the graphs in the training sets or should not be the subgraphs of the training set graphs. Moreover, the generated graphs should be diverse. Therefore, we compute the 9) *novelty*, which measures the percentage of generated graphs that are not subgraphs of the training graphs. We also use 10) *uniqueness*, which indicates the diversity in generated graphs, and 11) *neighborhood subgraph pairwise distance kernel* (NSPDK) [8], a valid metric for computing the global similarity of the generated graphs by considering both the structure and label of graphs.

Table 2
Statistics of the datasets used in our experiment.

Dataset	$\lvert\mathscr{G}\rvert$	$\lvert\mathscr{V}\rvert_{(min,max)}$	$\lvert\mathscr{E}\rvert_{(min,max)}$	$\lvert\mathscr{V}_L\rvert$	$\lvert\mathscr{E}_L\rvert$
Lung	24,836	[6, 50]	[5, 57]	11	3
Yeast	55,532	[6, 50]	[5, 57]	11	3
All	80,256	[5, 129]	[4, 135]	11	3
Enzymes	576	[2, 125]	[1, 149]	24	1
Citeseer-s	626	[333, 431]	[477, 736]	6	1
Cora-s	430	[346, 531]	[543, 760]	7	1

5.4. Experimental settings

For GraphRNN and GRAPHGEN, we replicate the methods and use the hyper-parameters adopted in the original studies. For the hyper-parameters of GHRNN, we refer to other RNN-based graph generative models, such as GraphRNN, DeepGMG, and GRAPHGEN. We test the hyper-parameters of our model within a reasonable range. For example, the batch sizes are usually 32, 64, 128, and 256. We test all batch sizes within [32] and select the best size. We consider Adam as our optimizer with a learning rate of 0.003, which is decayed by 0.3 at steps 100, 200, 400, and 800 in all experiments. The batch size on Yeast is set to 256, and the batch size on other datasets is set to 32 in our experiment. We split 80%, 10%, and 10% of the graphs from each dataset for training, validating, and testing, respectively. The training process stops when the validation loss reaches the minimum.

Because of the plain RNN's insensitiveness to long-term dependencies, we construct the edge-level RNN module based on the architecture of the LSTM. In recent years, several research fields, including semantic relation extraction [15], online incremental keyword extraction [11], and knowledge modeling [23], have generally adopted the LSTM as a basic framework. For GHRNN, the edge-level RNN consists of a four-layer LSTM with 256-dimensional hidden state in each layer. The dimension of input layer in the edge-level RNN module is set to 92. The element-level RNN module has a single layer of GRU cells with 256-dimensional hidden state. We also set the dimension of input layer in the element-level RNN module to 92. Additionally, each embedding function is fitted with an independent single-layer fully connected neural network. The output functions are two-layer MLPs.

For the two synthetic datasets described in Section 5.1, we set the restart probability of RWR to 0.15 and select a node to start RWR proportional to its degree. The sizes of the sampled subgraphs range from $333 \leqslant |\mathscr{V}| \leq 431$ and $477 \leqslant |\mathscr{E}| \leq 736$ on Citeseer-s and $346 \leqslant |\mathscr{V}| \leq 531$ and $543 \leqslant |\mathscr{E}| \leq 760$ on Cora-s.

All the experiments are performed on a Nvidia Geforce RTX 2080 Ti GPU card with 11 GB GPU memory and 1 TB RAM in an Ubuntu 18.04.1 operating system. Our model is implemented with PyTorch, a deep-learning framework. The code for implementing GHRNN is available at https://github.com/SXD-JLU/GHRNN.

5.5. Experimental results of learning the graph distributions

The comparison results of our framework and the baselines on six datasets are listed in Tables 3–5 and Fig. 6. To comprehensively compare the ability of each model on modeling graph distributions, we analyze the experimental results in terms of the following three perspectives: 1) the ability of learning the structural distributions of the datasets; 2) the ability of learning the label distributions of the datasets; 3) the novelty and uniqueness of the generated graphs.

Ability of Learning the Structural Distribution. In Table 3, the best performance achieved in terms of each metric for a particular dataset is highlighted in bold type and any value less than 10^{-3} is denoted as 0. All the subsequent tables follow these rules. As shown in Table 3, GHRNN achieves the best performance in learning the distribution of node degrees on all datasets, with a decrease of 20% and 64% in the MMD compared with GRAPHGEN and GraphRNN, respectively. Furthermore, GHRNN exhibits the best performance in learning the clustering coefficient distribution of nodes on all datasets, with a decrease of 14% and 65% in the MMD compared with GRAPHGEN and GraphRNN, respectively. In addition, GHRNN achieves the best performance in learning the orbit count distribution of nodes, with a decrease of 8% and 70% in the MMD compared with GRAPHGEN and GraphRNN, respectively. The experimental results on the three metrics show that GHRNN outperform GRAPHGEN and GraphRNN in capturing the structural information and distribution of graphs.

The upper and lower charts in Fig. 6(a) show the comparison in terms of the average number of nodes. The upper and lower charts in Fig. 6(b) present the comparison in terms of the average number of edges. Because the graphs in Cora-s and Citeseer-s are larger than those in other datasets, we use the upper and lower charts to display the experimental statistics on Cora-s a and Citeseer-s and other datasets, respectively. The captions on the left side of every sub-figure indicate the meaning of the vertical axis. In Fig. 6(a), the height of the bar in the charts denotes the average number of nodes on the datasets. In Fig. 6(b), the height of the bar in the charts denotes the average number of edges on the datasets. As shown in Fig. 6, GHRNN performs relatively worse on Citeseer-s and Cora-s. However, the average number of nodes and edges of graphs generated by GHRNN is closest to that of the original testing set of four other realistic datasets. These two metrics further reveal that GHRNN excels at modeling the structural distribution of graphs.

Table 3
Comparison in terms of graph structure.

	GHRNN			GRAPHGEN			GraphRNN		
	Degree	Cluster	Orbit	Degree	Cluster	Orbit	Degree	Cluster	Orbit
Lung	**0.006**	≈ **0**	≈ **0**	0.008	≈ **0**	≈ **0**	0.257	0.005	0.048
Yeast	**0.004**	≈ **0**	≈ **0**	0.006	≈ **0**	≈ **0**	0.206	0.314	0.015
All	**0.004**	≈ **0**	≈ **0**	**0.004**	≈ **0**	≈ **0**	0.185	0.056	0.018
Enzymes	**0.062**	**0.107**	**0.014**	0.221	0.163	0.022	0.191	0.155	0.113
Citeseer-s	**0.395**	**0.464**	**0.499**	0.494	0.602	0.645	0.469	0.57	0.704
Cora-s	**0.292**	**0.451**	0.852	0.3	0.612	**0.743**	0.302	0.559	0.899

Table 4
Comparison in terms of labels.

	GHRNN			GRAPHGEN			GraphRNN		
	Node Label	Edge Label	Joint	Node Label	Edge Label	Joint	Node Label	Edge Label	Joint
Lung	**0.001**	≈ **0**	**0.155**	**0.001**	≈ **0**	0.181	0.32	0.011	0.738
Yeast	≈ **0**	≈ **0**	**0.068**	≈ **0**	≈ **0**	0.103	0.258	0.019	0.761
All	0.001	≈ **0**	0.109	≈ **0**	≈ **0**	**0.098**	1.084	0.02	0.771
Enzymes	0.005	-	**0.106**	0.007	-	0.199	**0.003**	-	0.332
Citeseer-s	0.029	-	**0.305**	0.02	-	0.471	**0.001**	-	0.361
Cora-s	0.065	-	**0.105**	**0.048**	-	0.112	0.07	-	0.337

Table 5
Comparison in terms of similarity, novelty, and uniqueness.

	GHRNN			GRAPHGEN			GraphRNN		
	NSPDK	Nove.	Uniq.	NSPDK	Nove.	Uniq.	NSPDK	Nove.	Uniq.
Lung	**0.022**	99%	**100%**	0.031	**100%**	**100%**	0.308	**100%**	**100%**
Yeast	**0.018**	95%	**99%**	0.024	97%	**99%**	0.309	**100%**	98%
All	**0.018**	98%	99%	0.019	97%	99%	0.314	**100%**	**100%**
Enzymes	**0.064**	96%	97%	0.079	98%	**100%**	0.117	**99%**	86%
Citeseer-s	0.075	**100%**	**100%**	0.081	**100%**	**100%**	**0.069**	**100%**	**100%**
Cora-s	**0.076**	**100%**	**100%**	0.078	**100%**	**100%**	0.084	**100%**	**100%**

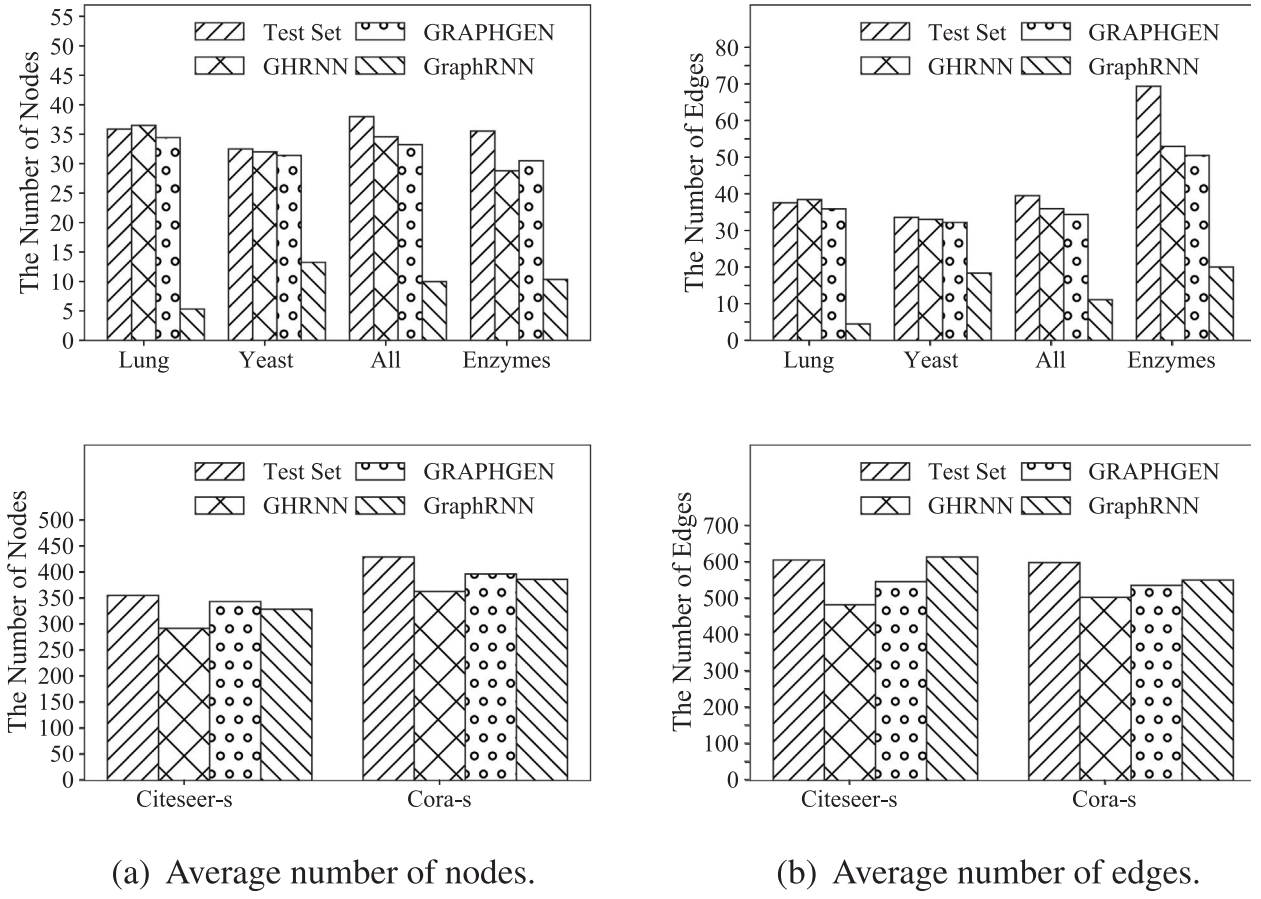


Fig. 6. Comparison in terms of the average number of nodes and edges.

Ability of Learning the Label Distribution. Table 4 summarizes the experimental results of capturing label distribution. Although GHRNN does not have a better capability to obtain the information of labels compared with the other two base-

lines, GHRNN achieves the best performance in learning the joint distribution of node labels and node degrees, with a decrease of 21% and 68% in the MMD compared with GRAPHGEN and GraphRNN, respectively. GHRNN's excellent performance on the joint distribution of node labels and node degrees establishes that GHRNN can comprehend the structural and label information of graphs simultaneously. These results indicate that GHRNN can learn the most realistic distribution of graph data compared with the other two methods.

Novelty and Uniqueness of Generated Graphs. Table 5 indicates that GHRNN achieves the best performance in terms of the NSPDK metric on almost all datasets, with a decrease of 15% and 55% in the MMD compared with GRAPHGEN and GraphRNN, respectively. The NSPDK metric can reflect the global similarity of generated graphs with test graphs. Therefore, the excellent performance in terms of the NSPDK metric demonstrates that GHRNN not only imitates the graph dataset but also learns the distribution of the graph dataset. In terms of the novelty and uniqueness, GHRNN yields at least 95% on all datasets, indicating that GHRNN can generate diverse graphs different from the graphs in the training set. Furthermore, to intuitively observe the quality of the generated graphs, we show the visualization of the graphs generated by each model and the graphs in the real datasets in Figs. 7 and 8.

Graph visualization. Fig. 7 shows molecular graphs randomly selected from the Lung dataset and the graphs generated by each model. In Fig. 7, we observe that the quality of the graphs generated by GHRNN and GRAPHGEN is significantly better than that of the graphs generated by GraphRNN, which exactly corresponds to the experimental results in the above tables. In addition, we find some incomplete benzene rings in Fig. 7(d), which may arise from GRAPHGEN's inability to learn the detailed associations between the elements. For large graphs, Fig. 8 illustrates that GHRNN can effectively capture the structural characteristics of graphs in Citeseer-s and Cora-s.

In summary, the comparison results indicate that all improvements are significant. The proposed framework, GHRNN, outperforms representative baseline methods for learning graph distributions, which answers the first question in this section.

5.6. Scalability analysis

Fig. 9 shows the scalability of models. Because some hyperparameters of each model differ during the training process, comparing the number of epochs is irrelevant. As we pay more attention to the time required for the model to complete the training process, the curve of the training time changing with the training process can more accurately reflect the scalability of the model. The training process is calculated as $(Loss_s - Loss_t)/(Loss_s - Loss_e) \times 100\%$. $Loss_s, Loss_t$, and $Loss_e$ represent the initial loss value, the current loss value, and the final loss value in the training process, respectively.

GRAPHGEN achieves the best performance in scalability among the domain-agnostic graph generative models. In Fig. 9, the training process of GHRNN is as fast as that of GRAPHGEN on the first five datasets. Furthermore, GHRNN outperforms GRAPHGEN in scalability on Enzymes. Although GHRNN performs worse relative to other methods on Cora-s, it can still complete the training process in a short time. In summary, GHRNN can achieve outstanding performance in scalability compared with baseline methods, which answers the second question asked in this section.

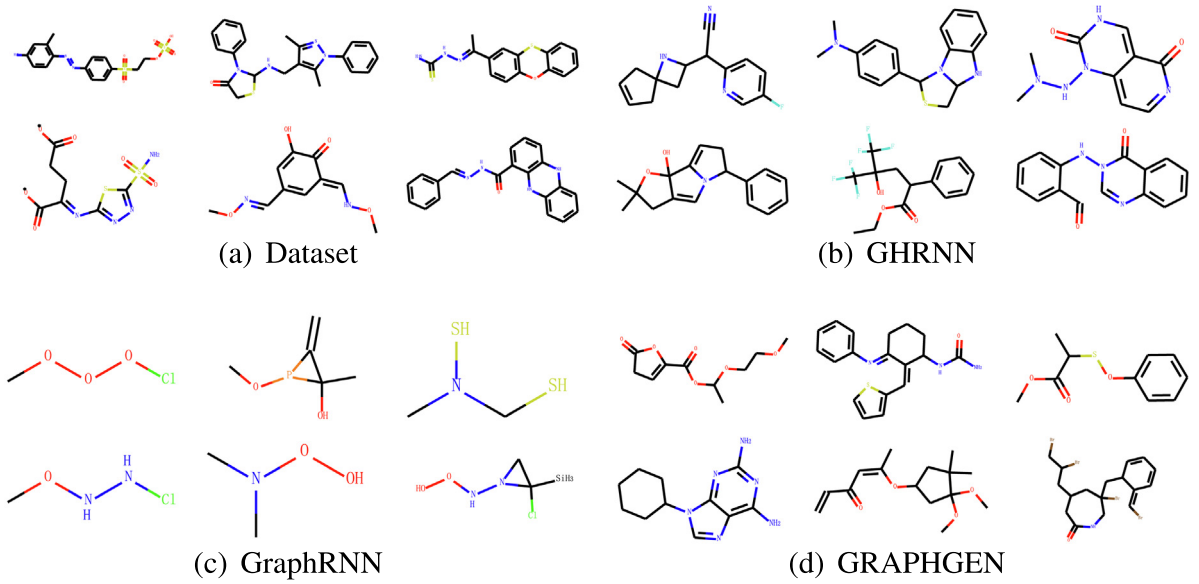


Fig. 7. Visual comparison of real molecules generated by different models.

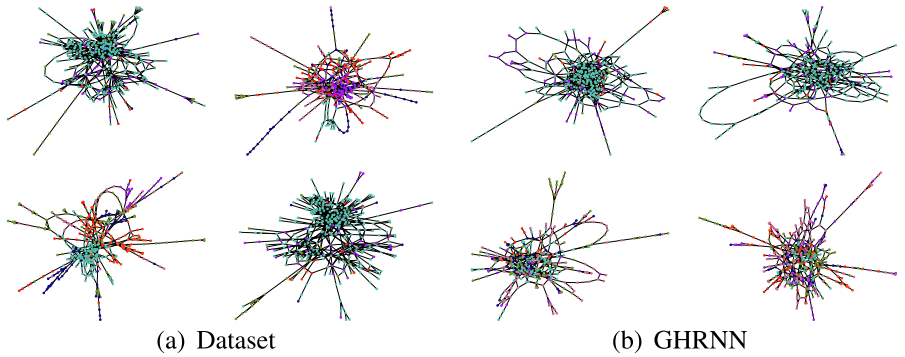


Fig. 8. Visual comparison of real graphs generated by GHRNN.

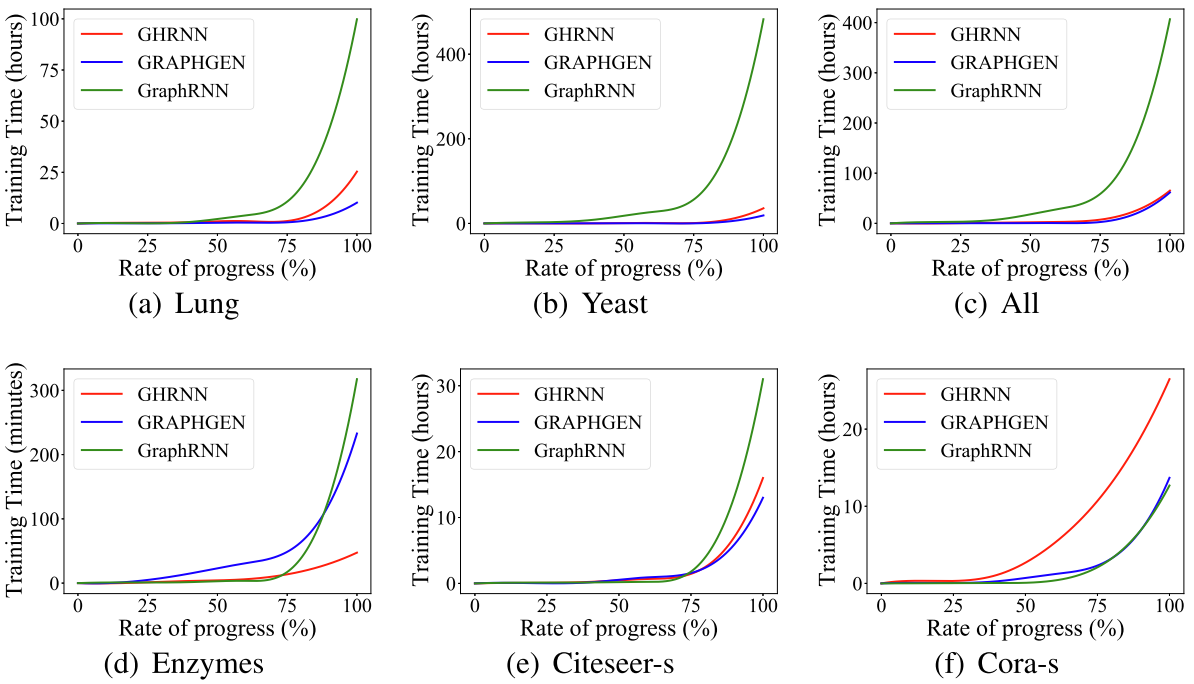


Fig. 9. Training curves of models on different datasets.

6. Conclusion and future work

In this paper, we propose GHRNN, which is a novel RNN-based model for graph generation. Our experimental results show that GHRNN can achieve the best performance in terms of most of the evaluation metrics. This indicates that the hierarchical RNNs are effective in graph generation. GHRNN can capture structural information and semantic information by employing the edge-level RNN module and the element-level RNN module, respectively; thus, it can better learn the probability distribution of the graphs. Furthermore, the reasonable comparison verifies that our model performs well on scalability.

Despite the considerable progress in graph generation models, there are still two problems to be solved. First, the deep graph generative models need a more reasonable training method, because the loss of supervised learning is not closely associated with the metrics for evaluating generated graphs. Second, to evaluate generated graphs more accurately, new appropriate metrics are required. In the future, we will investigate graph generation and the aforementioned problems. Specifically, we will attempt to generate molecules having better chemical properties such as drug-likeness and synthetic accessibility. Furthermore, we will try to obtain better metrics to evaluate the generated graphs.

CRediT authorship contribution statement

Song Xianduo: Conceptualization, Methodology, Software, Writing - original draft. **Wang Xin:** Validation, Writing - review & editing. **Song Yuyuan:** Data curation, Investigation. **Zuo Xianglin:** Visualization, Validation. **Wang Ying:** Supervision, Funding acquisition, Writing - review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work is supported by the Foundation of National Key R&D Program of China (No. 2021ZD0112502, 2021ZD0112501), the National Natural Science Foundation of China (No. 61872161), the Jilin Educational Committee (No. JJKH20191257KJ), the Nature Science Foundation of Jilin Province (20200201297JC), the Foundation of Development and Reform of Jilin Province (No. 2019C053-8), and the Interdisciplinary and Integrated Innovation of JLU (No. JLUXKJC2020207).

References

[1] United State National Library of Medicine National Center for Biotechnology Information. [n.d.], PubChem, URL: http://pubchem.ncbi.nlm.nih.gov..
[2] Edoardo M Airoldi, David M Blei, Stephen E Fienberg, Eric P Xing, Mixed membership stochastic blockmodels, Journal of Machine Learning Research (JMLR) 9 (9) (2008) 1981–2014.
[3] Réka Albert, Albert-László Barabási, Statistical mechanics of complex networks, Reviews of Modern Physics 74 (1) (2002) 47.
[4] Esben Jannik Bjerrum, Richard Threlfall, Molecular generation with recurrent neural networks (rnns). arXiv preprint arXiv:1705.04612, 2017..
[5] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, Stephan Günnemann, NetGAN: Generating graphs via random walks, in: Proceedings of the 35th International Conference on Machine Learning (ICML), 2018, pp. 610–619.
[6] Karsten M. Borgwardt, Cheng Soon Ong, Stefan Schönauer, S.V.N. Vishwanathan, Alex J. Smola, Hans-Peter Kriegel, Protein function prediction via graph kernels, Bioinformatics (Oxford, England) 21 (Suppl 1) (2005) 47–56..
[7] Jinyin Chen, Haibin Zheng, Hui Xiong, Shijing Shen, Su. Mengmeng, Mag-gan: Massive attack generator via gan, Information Sciences 536 (2020) 67–90.
[8] Fabrizio Costa, Kurt De Grave, Fast neighborhood subgraph pairwise distance kernel, in: Proceedings of the 26th International Conference on Machine Learning (ICML), 2010, pp. 255–262.
[9] Yuanfei Dai, Shiping Wang, Xing Chen, Chaoyang Xu, Wenzhong Guo, Generative adversarial networks based on wasserstein distance for knowledge graph embeddings, Knowledge-Based Systems (KBS) 190 (2020) 105165..
[10] Nicola De Cao and Thomas Kipf. MolGAN: An implicit generative model for small molecular graphs. arXiv e-prints, page arXiv:1805.11973, 2018..
[11] Xiaoyu Duan, Shi Ying, Hailong Cheng, Wanli Yuan, Xiang Yin, Oilog: An online incremental log keyword extraction approach based on mdp-lstm neural network, Information Systems (IS) 95 (2021) 101618..
[12] Paul Erdős, Alfréd Rényi, On the evolution of random graphs, Publ. Math. Inst. Hung. Acad. Sci 5 (1) (1960) 17–60.
[13] Fei Gao, Jingjie Zhu, Hanliang Jiang, Zhenxing Niu, Weidong Han, Yu. Jun, Incremental focal loss gans, Information Processing & Management (IPM) 57 (3) (2020) 102192.
[14] Wei Gao, Ayda Darvishan, Mohammad Toghani, Mohsen Mohammadi, Oveis Abedinia, Noradin Ghadimi, Different states of multi-block based forecast engine for price and load prediction, International Journal of Electrical Power & Energy Systems 104 (2019) 423–435.
[15] ZhiQiang Geng, GuoFei Chen, YongMing Han, Lu. Gang, Fang Li, Semantic relation extraction using sequential and tree-structured lstm with attention, Information Sciences 509 (2020) 183–192.
[16] Noradin Ghadimi, Adel Akbarimajd, Hossein Shayeghi, Oveis Abedinia, Two stage forecast engine with feature selection technique and improved meta-heuristic algorithm for electricity load forecasting, Energy 161 (2018) 130–142.
[17] Xavier Glorot, Yoshua Bengio, Understanding the difficulty of training deep feedforward neural networks, Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS) 9 (2010) 249–256.
[18] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, Yoshua Bengio, Generative adversarial nets, in: Advances in Neural Information Processing Systems 27 (NIPS), 2014, pp. 2672–2680..
[19] Nikhil Goyal, Harsh Vardhan Jain, Sayan Ranu, GraphGEN: A scalable approach to domain-agnostic labeled graph generation, in: Proceedings of The Web Conference 2020 (WWW), 2020, pp. 1253–1263.
[20] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, Alexander Smola, A kernel two-sample test, The Journal of Machine Learning Research 13 (1) (2012) 723–773.
[21] Gabriel Lima Guimaraes, Benjamin Sanchez-Lengeling, Carlos Outeiral, Pedro Luis Cunha Farias, Alán Aspuru-Guzik, Objective-reinforced generative adversarial networks (organ) for sequence generation models. arXiv e-prints, page arXiv:1705.10843, 2017..
[22] Sepp Hochreiter, Jürgen Schmidhuber, Long short-term memory, Neural Computation 9 (8) (1997) 1735–1780.
[23] Yujia Huo, Derek F. Wong, Lionel M. Ni, Lidia S. Chao, Jing Zhang, Knowledge modeling via contextualized representations for lstm-based personalized exercise recommendation, Information Sciences 523 (2020) 266–278.
[24] Wengong Jin, Regina Barzilay, Tommi Jaakkola, Junction tree variational autoencoder for molecular graph generation, in: Proceedings of the 35th International Conference on Machine Learning (ICML), 2018, pp. 2323–2332.
[25] Hossein Khodaei, Mahdi Hajiali, Ayda Darvishan, Mohammad Sepehr, Noradin Ghadimi, Fuzzy-based heat and power hub models for cost-emission operation of an industrial consumer using compromise programming, Applied Thermal Engineering 137 (2018) 395–405.
[26] Diederik P. Kingma, Max Welling, Auto-encoding variational bayes. arXiv e-prints, page arXiv:1312.6114, 2014..
[27] Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, Zoubin Ghahramani, Kronecker graphs: an approach to modeling networks, Journal of Machine Learning Research (JMLR) 11 (2) (2010) 985–1042.
[28] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, Peter Battaglia, Learning deep generative models of graphs, arXiv e-prints, page arXiv:1803.03324, 2018..
[29] Łukasz Maziarka, Agnieszka Pocha, Jan Kaczmarczyk, Krzysztof Rataj, Tomasz Danel, Michał Warchoł, Mol-cyclegan: a generative model for molecular optimization, Journal of Cheminformatics 12 (1) (2020) 1–18.
[30] D.E. Bidisha Samanta, Gourhari Jana Abir, Pratim Kumar Chattaraj, Niloy Ganguly, Manuel Gomez Rodriguez, NeVAE: A deep generative model for molecular graphs, in: Proceedings of the AAAI Conference on Artificial Intelligence, 2019, pp. 1110–1117.

[31] Martin Simonovsky, Nikos Komodakis, GraphVAE: Towards generation of small graphs using variational autoencoders, in: Proceedings of the 27th International Conference on Artificial Neural Networks (ICANN), Springer, 2018, pp. 412–422..
[32] Haozhen Situ, Zhimin He, Yuyi Wang, Lvzhou Li, Shenggen Zheng, Quantum generative adversarial network for generating discrete distribution, Information Sciences 538 (2020) 193–208.
[33] Cong Tran, Won-Yong Shin, Andreas Spitz, Michael Gertz, Deepnc: Deep generative network completion, arXiv preprint arXiv:1907.07381, 2019..
[34] David Weininger, Smiles, a chemical language and information system. 1. Introduction to methodology and encoding rules, Journal of Chemical Information and Computer Sciences 28 (1) (1988) 31–36.
[35] Wu. Lianwei, Yuan Rao, Ambreen Nazir, Haolin Jin, Discovering differential features: Adversarial learning for information credibility evaluation, Information Sciences 516 (2020) 453–473.
[36] Xifeng Yan, G.S.P.A.N. Jiawei Han, Graph-based substructure pattern mining, in: Proceedings of 2002 IEEE International Conference on Data Mining (ICDM), 2002, pp. 721–724.
[37] Carl Yang, Peiye Zhuang, Wenhan Shi, Alan Luu, Pan Li, Conditional structure generation through graph variational generative adversarial nets, in: Proceedings of the 33th Advances in Neural Information Processing Systems (NIPS), 2019, pp. 1340–1351.
[38] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, Jure Leskovec, GraphRNN: Generating realistic graphs with deep auto-regressive models, in: Proceedings of the 35th International Conference on Machine Learning (ICML), 2018, pp. 5708–5717..
[39] Yu. Lantao, Weinan Zhang, Jun Wang, Yu. Yong, SeqGAN: Sequence generative adversarial nets with policy gradient, in: Proceedings of the 31th AAAI Conference on Artificial Intelligence, 2017, pp. 2852–2858.
[40] Xingwang Zhao, Jiye Liang, Jie Wang, A community detection algorithm based on graph compression for large-scale social networks, Information Sciences 551 (2021) 358–372.
[41] Jun-Yan Zhu, Taesung Park, Phillip Isola, Alexei A Efros, Unpaired image-to-image translation using cycle-consistent adversarial networks, in: Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2223–2232.