ForGen: Autoregressive Generation of Sparse Graphs with Preferential Forest

Yao Shi[1], Yu Liu[1], and Lei Zou[1]

Peking University, Beijing, China
{shall, dokiliu, zoulei}@pku.edu.cn

Abstract. Graph is a natural way to model interactions between objects, such as in the field of biology and social science. Over the past decades, modeling and generating graphs have been a popular research topic, largely inspired by observed properties of real-world graphs. Since traditional approaches relying on hand-crafted mechanisms are only capable of capturing some specific graph properties, recent focus has been shifted to deep neural methods. However, the task is still challenging in terms of efficiency. To address this issue, we observe that the connectivity (i.e., degree) of nodes follows scale-free distribution for most real-world graphs, which can be utilized to accelerate the generation process. We propose ForGen, a Forest-based Generation model that contains a graph-level and an edge-level autoregressive generator. Specifically, for the edge-level model, motivated by the skewed distribution of node degree and the Huffman tree, we design a forest-like data structure to accelerate edge connection via shallow tree searches and better parallelism. Experiments on both synthetic and real-world graph datasets show that ForGen is two times faster than the current state-of-the-art method for graph generation, and guarantees better generated graph quality.

Keywords: Graph Generation · Autoregressive Model · Neural Network.

1 Introduction

Graphs are ubiquitous in the real world, which can represent relational information such as social networks, traffic networks, knowledge bases and molecule structures. To tap the underlying value of complicated structures in these graphs [2], there are mainly two types of approaches [9]: 1) predicting and analyzing patterns on given graphs. 2) modeling and generating graphs, while we focus on the second one.

Since the early work by Erdos and Rényi in 1960 [5], many graph generative models have been proposed due to its wide range of applications which includes mining patterns in social networks [1,8,15], drug design [17,29,22], knowledge graph completion [27], and architecture search [28]. Traditional generative models for graphs (e.g. Barabási-Albert model [1], Kronecker graphs [15], small-world networks [26], and stochastic block models [23]) are based on prior assumptions and random graph theory, which usually have clear mathematical properties and computational efficiency but lack the ability to model complex dependencies due to their hand-crafted nature. Also, prior assumptions limit the generalization ability and versatility of the model.

To tackle these limitations, recent studies have turned to deep generative models using neural networks. Instead of making prior assumptions, deep graph generators directly learn the distribution from given data to capture latent structural information. Despite the great progress achieved in the molecular area (graph size is small), modeling graphs in the general domain has specific challenges. As a graph with n nodes has an n^2 sized adjacency matrix, most non-autoregressive models [22,29] trying to generate the full matrix fail to scale up to graphs with hundreds of nodes. Moreover, methods like autoencoder-based graph generators [22] treat a graph as a combination of independent components or structures with weak dependency to speed up the generation process. However, this assumption ignores the complex structure dependencies in real-world graphs which can sacrifice the quality of the generated graphs [18].

For the reasons mentioned above, autoregressive models have gained special attention for modeling general real-world graphs (not just molecules) [30,18,7,4], whose basic idea is to sequentially add nodes and edges or small motifs into the graph. Thus new decisions are made based on previous information and complex dependencies can be introduced into the model. Generally, there are two types of methods according to the generation framework they choose. The first way is to use hierarchical architectures, for instance, GraphRNN [30] adopts a graph level RNN combined with an edge level RNN. Another type of approaches leverages massage passing mechanism to bring more neighborhood information into the process [18]. However, the lack of features (semantic information) in the problem setting may hinder GNN's performance and the scalability of GNN and simple sequential models can also be a bottleneck of efficiency.

So far, BiGG [4] is the most scalable autoregressive graph generator and can achieve state-of-the-art sample quality in both synthetic and real-world datasets. Based on the observation that most real graphs are sparse, BiGG attempts to generate the non-zero entries in the adjacency matrix instead of the whole matrix. As GraphRNN does, BiGG adopts a two-level (node and edge) architecture but it reduces the time complexity from $O(n^2)$ to $O((n+m)\log n)$ for a graph with n nodes and m edges. More specifically, BiGG uses a binary tree data structure as the edge level to reduce the time complexity of generating an edge from $O(n)$ to $O(\log n)$, which is the key to efficiency improvement.

It is well studied that the vertex connectivities commonly follow a scale-free distribution in complex real networks [21,3,20]. However, we observed that existing studies have not paid close attention to this characteristic in both experimental setup and model design. As a matter of fact, the degree distributions of graph datasets used in previous work (such as grid graphs and protein graphs [18,4]) are more close to the normal distribution rather than the scale-free distribution, which does not match most real-world application scenarios. Through experiments, we found that it is harder for deep models to learn from scale-free networks than other graph datasets under the same settings, which indicates their complex latent patterns.

When it comes to model design, BiGG treats all the nodes in the same way during the edge generation process. Nevertheless, it is well-known that node degree of numerous real-world graphs (approximately) follows heavy tailed (e.g., scale-free) distribution, and exhibits interesting phenomenons such as "rich-get-richer". This indicates that one should treats nodes *differently* in generating graphs. To make a further improvement on the two-level autoregressive framework, we propose ForGen (Forest-based Generation Model) to model these propeties explicitly. Instead of a balanced tree, ForGen conducts the edge level as a hierarchical forest-like data structure inspired by the Huffman tree [10] and the "rich-get-richer" phenomenon, which enables shallow tree search. Also, we improve the traversal-based learning algorithm used in the edge level for better scalability and quality. Thereby, we can generate an edge in $O(1)$ time for the best case which has a higher frequency, and $O(\log n)$ time for the worst case.

To summarize, we have made the following contributions:

- To the best of our knowledge, it is the first time that the skewed distribution of node degree in real-world graphs has been considered in graph generation with deep neural network.
- We propose a two-level deep autoregressive model ForGen with an imbalanced forest-like structure for graph generation, which exploits the "rich-get-richer" phenomenon in scale-free networks.
- Empirical evaluation on both scale-free and non-scale-free networks establishes that compared with the state-of-the-art technique, ForGen can achieve better quality on four datasets and a twice faster sample speed.

2 Related Work

In this section, we will briefly introduce the existing methods in three parts. Table 1 summarizes several notable deep graph generators and their characteristics.

Traditional Graph Models. In the Erdos Rényi random graph model [5], each edge is included with a fixed probability p independently and the degree distribution is proved to be Poisson for large n and $p = c/n$, where c is a constant. However, Barabási et al. [3] find that the degree distribution is commonly a scale-free distribution for realistic networks and propose the Barabási-Albert (BA) model based on the preferential attachment assumption. Though there is various work following the preferential attachment model [1,12,16], they only focus on the degree distribution while neglecting other structural features.

Autoregressive Deep Graph Generators. Li et al. [17] propose a node-by-node method that includes two variants, namely MolMP and MolRNN, to model the probabilities for adding nodes and termination. Similarly, You et al. [30] propose GraphRNN based on two RNNs (one for graph level and one for edge level). Subsequently, Liu et al. [19] combine GraphRNN with a vanilla

Table 1. Existing deep graph generators. n and m denote the number of nodes and edges respectively. M is calculated from the data.

Method	Technology Roadmap	Complexity
GraphVAE [22]	Auto-encoder (GNN + MLP)	$O(n^4)$
GCPN [29]	Reinforcement Learning	$O(m^2)$
Graphite [8]	Auto-encoder + GNN	$O(n^2)$
GraphRNN [30]	Autoregressive-based + Two level	$O(nM)$
GRAN [18]	Autoregressive-based + GNN	$O((m+n)n)$
BiGG [4]	Autoregressive-based + Two level	$O((m+n)\log n)$
ForGen (ours)	Autoregressive-based + Two level	$O((m+n)\log n)$

Transformer decoder. Liao et al. [18] propose a GNN-based approach that uses graph neural network with attentive massages to generate one block of nodes and associated edges at each step. Apart from node-based methods, Goyal et al. [7] adopt LSTM to generate edge sequences which are converted from a graph using the minimum DFS code. Recently, Dai et al. [4] propose BiGG, a two-level model with tree-based architecture which achieves the best performance in both quality and scalability.

Non-autoregressive Deep Graph Generators. There exist various work that focuses on generating graphs in a non-autoregressive way. For example, GraphVAE [22] constructs its encoder and decoder based on GCN and MLP respectively, which needs a costly graph matching algorithm to train. Graphite [8] parameterizes variational autoencoders with a GNN, and uses an iterative graph refinement strategy inspired by low-rank approximations for decoding, which can only learn from one input graph. You et al. [29] formulate the molecular graph generation as a Markov Decision Process and adopt PPO to train an RL agent. However, these non-autoregressive graph generators are mainly designed for molecular generation which may suffer from the scalability issue and lack the ability to model complex dependencies.

3 Methodology

3.1 Preliminaries

Graph Generation. A graph is defined by the tuple $G = (V, E)$, where $V = \{v_1, ..., v_n\}$ is the graph's node set and $E = \{(v_i, v_j)|v_i, v_j \in V\}$ denotes its edge set, with $|V| = n$ and $|E| = m$. We represent a graph with an adjacency matrix $A \in \{0, 1\}^{n\times n}$, while a graph has up to $n!$ different matrix representations with an ordering set $\mathit{\Pi}$ over the nodes. More precisely, given an ordering $\pi \in \mathit{\Pi}$, $(\pi(v_1), ..., \pi(v_n))$ is a permutation of $(v_1, ..., v_n)$ and a graph is mapped to a matrix A^π. Here, we focus on undirected graphs with no self-loops or attributes.

Our goal is to learn the distribution $p_{model}(\mathbb{G})$ from a given graph set $\mathbb{G} = \{G_1, ..., G_N\}$. Assuming that the observed graphs $\mathbb{G}$ are sampled from an underlying data distribution $p_{data}(\mathbb{G})$, an effective model should be capable of sampling graphs from the learned distribution which are similar to the real ones, i.e., $p_{model}(\mathbb{G}) \approx p_{data}(\mathbb{G})$. Therefore, we maximize the log-likelihood of observed graphs. Following GRAN [18], we use a single canonical ordering $\pi(G)$ to get a valid lower bound of $\log p(G)$ and significantly improve training efficiency:

$$\log p(G) = \sum_{\pi \in \Pi} \log p(A^{\pi}) \simeq \sum_{\pi \in \Theta \subset \Pi} \log p(A^{\pi}) \simeq \log p(A^{\pi(G)}) \tag{1}$$

Table 2. Main notations used in the paper.

Notation	Description
G, V, E	A graph, its node set, its edge set.
n, m	The number of nodes and edges respectively.
π	A node ordering for a graph G.
A^{π}	The adjacency matrix under π.
S^{π}_t	The t-th row of A^{π}'s lower triangle part.
$L(t)$	The number of trees for node v_t.
b	The depth of the first tree.
$\mathbf{x}$	A learnable vector $\in \mathbf{R}^d$.
$\mathbf{g}$, $\mathbf{h}$	A tuple of embedding vectors $\in \mathbf{R}^d$.

Important Characteristics of Real-world Graphs. The heavy-tailed degree distribution in real networks is first found by [21], i.e., the number of nodes of degree k being in inverse proportion to k^{γ} ($\gamma \in (1, \infty)$). The class of networks exhibiting a scale-free degree distribution (at least asymptotically) is called "scale-free network". Moreover, in real-world graphs, there is a higher probability that more and more nodes will link themselves to those with large degrees. In another word, some nodes (called hubs) can have much more connections than others and the existence of this mechanism (denoted as "Preferential Attachment" [3]) has been proved in many real-world graphs [11,14].

3.2 Overview

As shown in Fig 1, ForGen is a two-level autoregressive model based on tree structures. The iterative generation process consists of a top-down decoding stage and a bottom-up encoding stage. The decoding stage starts from the graph level which provides the initial embedding for a new node. To leverage the skewed degree distribution of real graphs, inspired by the preferential attachment, an imbalanced forest-like structure is used as the edge level. After the non-zero entries of A^{π} are decoded from the embedding, new edges are generated. Then the bottom-up stage encodes the edges into a vector within the edge level, which is utilized to update the graph level information. As shown in Table 2, we use $\mathbf{h}$ to represent the embeddings in the decoding stage and $\mathbf{g}$ in the encoding stage.

Formally, We represent the t-th row of A^{π}'s lower triangle part as S^{π}_t and the graph level can be factorized in an autoregressive way as

$$p(A^{\pi}) = \prod_{t=1}^{n} p(S^{\pi}_t | S^{\pi}_1, ..., S^{\pi}_{t-1}) \tag{2}$$

For edge level, instead of generating edges sequentially [30,4] or all at once [18], we divide S_t^π into $L(t)$ groups $S_{t,j}^\pi$ and organize them in a forest-like way:

$$p(S_t^\pi | S_1^\pi, ..., S_{t-1}^\pi) = \prod_{j=1}^{L(t)} p(S_{t,j}^\pi | S_{t,1}^\pi, ..., S_{t,j-1}^\pi, S_1^\pi, ..., S_{t-1}^\pi) \tag{3}$$

Based on this assumption, we generate the non-zero entries inside each group which can introduce more dependencies between groups into the model and take advantage of node diversity in real-world graphs to improve efficiency.

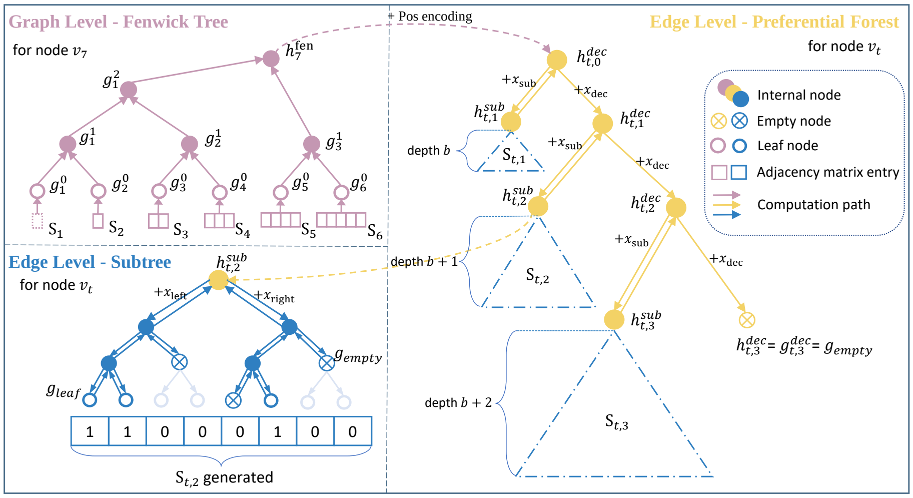
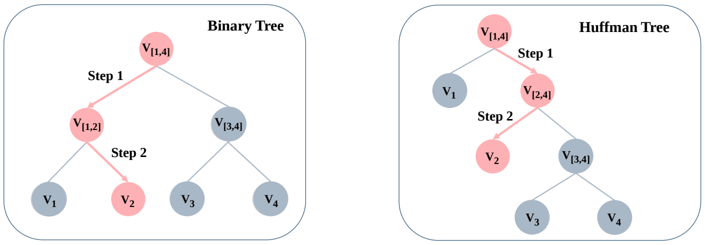


Fig. 1. An example of ForGen's generation process. The generation starts from the graph level computation (block on the top left) and goes to edge level (right part) following the dotted line. Edge level's Preferential forest is modeled in an inorder traversal way, while the subtrees of the forest (the blue part) are built from top to bottom and summarize the generated edges from leaves to the root.

3.3 A Hierarchical Tree Structure for Edge Generation

The edge level of ForGen is responsible for generating all the edges between node v_t and nodes already generated at timestamp t in an autoregressive way, which is equivalent to finding v_t's neighbors $\mathcal{N}(v_t)$ from a subset of nodes $\{v_1, ..., v_{t-1}\}$. Compared with the graph level, this stage is more critical to improving model efficiency. In this section, we will describe ForGen's edge generation strategy.

Generating with a Standard Binary Tree. BiGG [4] modeled the edge level as an edge binary tree. The binary tree is balanced for the reason that the maximum height difference between the left and right subtrees of each node is one. As illustrated in Fig 2(a), each leaf node represents a candidate for v_t's neighbors and the selection process is decomposed into a series of left/right decisions until reaching the bottom. However, in such a model structure, all the nodes are treated equally which contradicts the nature of real networks.



(a) (b)

Fig. 2. Simplified edge generation process for node v_5.

Edge Generation Considering Preferential Attachment. Instead of a balanced binary tree, a tree structure that takes nodes' access frequencies into account, will provide a better fit to the scale-free nature of real-world networks. Inspired by the Huffman tree [10], a frequency-sorted binary tree, we organized the tree in almost a sequential way. As Fig. 2(b) shows, all the right children are leaf nodes and represent the candidates. Thereby, the distance between hubs (nodes with large degrees) and the root will be $O(1)$ instead of $O(\log_2 n)$. Since hubs will be chosen frequently, it can significantly improve efficiency. We call it Preferential Tree since nodes closer to the root are preferentially treated.

However, there are many nodes far from the root. Though they may have much lower visiting frequency than the hubs, the structure can be optimized to avoid the $O(n)$ tree depth. Here, we introduce Preferential Forest, which replaces the leaf nodes with binary trees and all the trees make up a forest.

More specifically, the whole Preferential Forest built for node v_t represents the corresponding row S_t^π, and each tree in the forest (denoted as $S_{t,j}^\pi$) further separates S_t^π into different parts. Noted that nodes with higher degrees are fewer and there exists communities of distinct sizes in real graphs, we should keep tree sizes different. We define the j-th tree as the one whose root node is in the j-th level of the whole preferential forest, and its depth is $b + j - 1$, where b is a hyperparameter and $j \in \{1, 2, ..., L(t)\}$. Therefore, the size of the j-th group $|S_{t,j}^\pi| = 2^{b+j-1}$, which means the number of hub nodes are small and there exists many nodes with small degrees.

Lemma 1. *Considering the generation of node t's neighbors, the number of trees in the preferential forest* $L(t) = \lceil \log_2 (t + 2^b - 1) - b \rceil = O(\log n)$.

Proof. The number of nodes in $L(t)$ trees is $\sum_{j=1}^{L(t)} 2^{b+j-1} = 2^{b+L(t)} - 2^b$. Since the last tree may be filled with some virtual nodes, we have $\sum_{j=1}^{L(t)-1} 2^{b+j-1} < t - 1 \leq \sum_{j=1}^{L(t)-1} 2^{b+j-1}$. Therefore, $L(t) = \lceil \log_2 (t + 2^b - 1) - b \rceil = O(\log n)$.

The groups are generated sequentially based on Eq. 3 and the tree of each group is generated recursively until reaching the leaf node.

Theorem 1. *In a graph with* n *nodes, the time complexity of generating one edge with ForGen is* $O(1)$ *in the best case and* $O(\log n)$ *in the worst case.*

Proof. The process consists of two parts, i.e., reaching a tree in the preferential forest and traveling from the tree root to a leaf node. In the best case, the first stage costs only one step regardless of n and the tree depth is a constant b. So the time complexity is $O(1+b) = O(1)$. In the worst scenario, we will go to the last tree, which is the largest one, with $L(n)$ steps and the corresponding tree depth is $b+L(n)-1$. Accordingly, the time complexity is $O(2L(n)+b-1) = O(\log_2 n)$.

Generating $\boldsymbol{\mathcal{N}(v_t)}$ Simultaneously. As the generating process of an edge can be represented as a path from the root node to a leaf node, we can merge all the paths belonging to v_t to utilize their overlapping parts for overhead reduction. In other words, we convert the generation of $\mathcal{N}(v_t)$ into pruning v_t's preferential forest so that only the path to its neighbors' leaves are retained.

3.4 Two-level Autoregressive Model

In this section, we will present the full version of ForGen in detail (see Fig. 1). First, we will discuss how to embed a new node v_t at graph level. Then we will decode its embedding into $\mathcal{N}(v_t)$ via massage passing on the preferential forest.

Generating Initial Embedding $\boldsymbol{h_t}$ for $\boldsymbol{v_t}$. Following BiGG, we adopt the Fenwick tree [6], a data structure that maintains the prefix sum efficiently, as the graph level. The i-th leaf node of the tree $\mathbf{g}_i^0$ is the embedding of S_i generated before. With a Binary TreeLSTM cell [24], the tree is built from bottom to top and the i-th node in the j-th level

$$\mathbf{g}_i^j = \text{TreeLSTM}^{\text{fen}}(\mathbf{g}_{i*2-1}^{j-1}, \mathbf{g}_{i*2}^{j-1}) \tag{4}$$

Thus, we can obtain v_t's initial embedding $\mathbf{h}_t$ by computing the prefix sum of $\{\mathbf{g}_1^0, ..., \mathbf{g}_{t-1}^0\}$ with LSTM (go to the top left part of Fig. 1 for an example):

$$\mathbf{h}_t^{fen} = \text{LSTM}^{\text{fen}}([\mathbf{g}^k_{\lfloor \frac{t-1}{2^k} \rfloor}, \text{where } (t-1) \,\&\, 2^k = 2^k]) \tag{5}$$

Note that & is bitwise AND, the sequence length will be $O(\log n)$ and so is the time complexity of a graph level update.

Generating Preferential Forest. For this part, we need to generate edges between v_t and $\{v_1, ..., v_{t-1}\}$ based on the prefix sum result of the Fenwick tree and then encode the edges into a new embedding $\mathbf{g}_t^0$ to update the graph level.

As previously described in 3.3, the preferential forest consists of several binary subtrees which are always the left children (yellow part in Fig. 1), and the right children maintain the information of the generated graph. The root node $\mathbf{h}_{t,0}^{dec}$ is the sum of $\mathbf{h}_t^{fen}$ and a positional encoding which is the same as [25] to identify different steps and improve generation quality:

$$\mathbf{h}_{t,0}^{dec} = \begin{cases} \mathbf{h}_t^{fen} + \sin((\mathrm{n}-\mathrm{t})/10000^{\mathrm{i}/\mathrm{d}}), i \bmod 2 = 0 \\ \mathbf{h}_t^{fen} + \cos((\mathrm{n}-\mathrm{t})/10000^{(\mathrm{i}-1)/\mathrm{d}}), i \bmod 2 = 1 \end{cases} \tag{6}$$

where the dimension $i \in \{0, 1, ..., d-1\}$.

Then we decode the information by inorder traversal using TreeLSTM and LSTM. First, we need to obtain the root embedding for the subtree in the next level and predict if it has children via a Bernoulli distribution:

$$\mathbf{h}^{sub}_{t,i} = \mathrm{LSTMcell}^{\mathrm{toSub}}(\mathbf{h}^{dec}_{t,i-1}, \mathbf{x}_{sub}) \tag{7}$$

$$p(has\_subtree|\mathbf{h}^{sub}_{t,i}) = \sigma(\mathrm{MLP}^{\mathrm{sub}}(\mathbf{h}^{sub}_{t,i})) \tag{8}$$

where $\mathbf{x}_{sub}$ is a learnable vector to integrate direction information and σ is the sigmoid function.

Based on the $\mathbf{h}_{t,i}$, we need to generate the binary subtree before going back to the parent node. However, instead of a traversal way, we build the tree with a bisection method to significantly reduce the sequence length. For the reason that the forest can introduce more complex interactions between node groups, such a partial autoregressive way will not influence the effectiveness but can improve the LSTM-based model's scalability. The process can be represented as follows:

$$p(has\_pos|\mathbf{h}^{sub}_{u}) = \sigma(\mathrm{MLP}^{\mathrm{pos}}(\mathbf{h}^{sub}_{u})) \tag{9}$$

$$\mathbf{h}^{sub}_{\tau(u,pos)} = \mathrm{LSTMcell}^{\mathrm{sub}}(\mathbf{h}^{sub}_{u}, \mathbf{x}_{pos}) \tag{10}$$

where $pos \in \{left, right\}$ and $\tau(u, pos)$ means the *pos* child of u in the subtree. We will decide whether a child is an empty node based on the probability This process will continue recursively until no non-empty children or reaching the leaf node where we will add an edge between v_t and the corresponding node.

After all leaves of the subtree are generated, we need to summarize the neighborhood information from bottom to top. Thus, we represent leaf nodes and empty nodes with learnable parameters $\mathbf{g}_{leaf}$ and $\mathbf{g}_{empty}$ respectively and the bottom-top merging follows the rule:

$$\mathbf{g}^{sub}_{u} = \mathrm{TreeLSTMcell}^{\mathrm{sub}}(\mathbf{g}^{sub}_{\tau(u,left)}, \mathbf{g}^{sub}_{\tau(u,right)}) \tag{11}$$

Given $\mathbf{g}^{sub}_{t,i}$, we can complete the traversal of the preferential forest:

$$\mathbf{h}^{tmp}_{t,i} = \mathrm{TreeLSTMcell}^{\mathrm{dec}}(\mathbf{h}^{sub}_{t,i}, \mathbf{g}^{sub}_{t,i}) \tag{12}$$

$$\mathbf{h}^{dec}_{t,i} = \mathrm{LSTMcell}^{\mathrm{dec}}(\mathbf{h}^{tmp}_{t,i}, \mathbf{x}_{dec}) \tag{13}$$

When it goes to the empty node $\mathbf{h}^{dec}_{t,L(t)}$, all subtrees $S_{t,j}$ are generated which means S_t is available. To encode S_t, we adopt a bottom-top method:

$$\mathbf{g}^{dec}_{t,i} = \mathrm{TreeLSTMcell}^{\mathrm{forest}}(\mathbf{g}^{sub}_{t,i+1}, \mathbf{g}^{dec}_{t,i+1}) \tag{14}$$

where $\mathbf{g}^{dec}_{t,L(t)} = \mathbf{g}_{empty}$ and $i \in \{0, 1, ..., L(t) - 1\}$.

The final result $\mathbf{g}^{0}_{t} = \mathbf{g}^{dec}_{t,0}$ will be added to the Fenwick tree so as to generate the next node v_{t+1} starting from the graph level.

Training Optimization. In previous sections, we describe the inference stage of ForGen. However, if we train the model based on the full $O(n \log n)$ steps as the inference stage does, it will be too time-consuming. Since we have complete adjacency matrix data, we can prepare the training targets concurrently rather than step-by-step. The overall flow path is bottom-top-bottom. First, we gather all subtrees of graphs in the batch and align them from leaf nodes so that subtrees can be built synchronously. Then we present every preferential forest as a sequence and process them together starting from the empty node. After that, all node embeddings based on ground truth are ready with $O(\log n)$ steps and the top-bottom part can be done in a reverse procedure.

4 Experiments

In this section, we compare ForGen to state-of-the-art baselines on both synthetic and real graph datasets. The empirical results demonstrate the effectiveness and efficiency of ForGen.

4.1 Datasets

To make a more comprehensive comparison, we experiment on both scale-free and non-scale-free networks, and part of the benchmark is obtained from [4]. The description of the four datasets is as follows: (1) Grid: standard 2D grid graphs with $100 \leq |V| \leq 400$. (2) BA: random graphs using Barabási–Albert preferential attachment [3] which exhibit a clear scale-free distribution, with $50 \leq |V| \leq 150$. We experiment with different graph densities for further verification. (3) Youtube: 2-hop ego graphs extracted from the social network of YouTube users and their friendship connections [13] with $305 \leq |V| \leq 477$ and $684 \leq |E| \leq 5674$. (4) Amazon: 2-hop ego graphs extracted from the co-purchase network of Amazon [13] with $304 \leq |V| \leq 476$ and $510 \leq |E| \leq 1377$. Each dataset consists of 100 graphs of which 80 for training and 20 for test.

4.2 Settings

Baselines. Following previous work, we adopt the Erdos-Renyi random graph model [5] for traditional approaches and its parameter of graph density is estimated from the training set. For deep graph generators, we compare with GraphRNN[1], GRAN[2] and BiGG[3]. GraphRNN uses a two-level LSTM model while GRAN incorporates GNN with attention mechanism. BiGG is the state-of-the-art model which is designed for sparse graphs. For all the baselines, we use the original code released by their authors.

Evaluation Metrics. We use the same metric as proposed by [30], which is also used in [18,4,7]. Specifically, we use three graph statistics to describe the graph in different levels, i.e., node degree distribution, clustering coefficient distribution of nodes, and orbit count distribution (the number of occurrences of all orbits with 4 nodes). To compare the distribution between the graph statistics of generated graphs and test graphs, we use the Maximum Mean Discrepancy (MMD) with total variation (TV) distance.

[1] https://github.com/snap-stanford/GraphRNN

[2] https://github.com/lrjconan/GRAN

[3] https://github.com/google-research/google-research/tree/master/bigg

4.3 Model Effectiveness

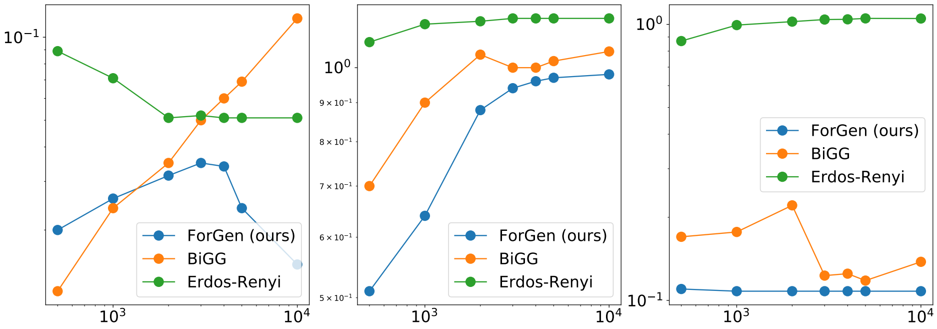
Generation Quality. The first experiment compares the sample quality between ForGen and other baselines across three different metrics on the four datasets. Table 3 shows the experiment results and the best performance under each metric-dataset pair is highlighted in bold font. On the whole, our method, ForGen, outperforms other methods over 3 metrics except for the clustering coefficient of BA dataset which is still the second-best. Also, the result of grid graphs shows that our method can be extended to non-scale-free graphs and achieve good performance.

Table 3. Performance comparison of different methods on the two synthetic datasets and two real-world datasets. For all MMD metrics, the smaller the better.

Datasets		Methods				
		Erdos-Renyi	GraphRNN	GRAN	BiGG	ForGen (ours)
Grid	Degree	0.79	$1.12e^{-2}$	$8.23e^{-4}$	$4.00e^{-4}$	$\mathbf{1.74}e^{-4}$
	Clustering	2.00	$7.73e^{-5}$	$3.79e^{-3}$	$7.00e^{-5}$	**0**
	Orbit	1.08	$1.03e^{-3}$	$1.59e^{-3}$	$5.00e^{-4}$	$\mathbf{2.23}e^{-4}$
BA	Degree	0.17	$9.33e^{-2}$	$1.73e^{-2}$	$1.42e^{-3}$	$\mathbf{1.25}e^{-3}$
	Clustering	0.53	0.31	0.39	**0.14**	0.15
	Orbit	1.16	0.70	0.30	0.36	**0.20**
Youtube	Degree	0.39	$1.60e^{-2}$	$4.40e^{-2}$	$1.30e^{-2}$	$\mathbf{5.84}e^{-3}$
	Clustering	0.75	0.33	0.15	0.12	**0.11**
	Orbit	0.22	0.11	**0.10**	**0.10**	**0.10**
Amazon	Degree	$8.61e^{-2}$	$1.13e^{-2}$	$1.18e^{-2}$	$1.09e^{-2}$	$\mathbf{1.01}e^{-3}$
	Clustering	1.00	0.48	0.18	0.37	**0.11**
	Orbit	0.93	0.13	0.11	$9.83e^{-2}$	$\mathbf{9.57}e^{-2}$

Sample Quality vs. Graph Scale. As shown in Table 3, BiGG (better than other baselines in most cases) performs close to our method in some instances. For further comparison, we test the sample quality from another aspect. Note that it is quite difficult to train on datasets of large graphs due to the limitation of GPU memory, model's ability of expansion is important. To be specific, we will sample graphs of increasing sizes with models trained on the previously mentioned datasets. As such, model performance may diverge and different characteristics of models are amplified. What's more, it offers us an efficient way to get larger graphs with structural information maintained.

However, GraphRNN stops the generation process based on its output which cannot be intervened manually and GRAN can only generate graphs no larger than their training graphs. So, only Erdos-Renyi model, BiGG and ForGen are included in the test based on the Amazon dataset. Fig. 3 shows the log-log plot of the MMD results on three graph statistics with graph sizes ranging from 0.5k to 10k. Notably, BiGG's performance on degree distribution declines continuously as the graph size grows while the other two methods are more stable. For clustering coefficient and orbit count, ForGen has consistently lower MMD than BiGG, and ER model is the worst.



(a) Degree (b) Cluster (c) Orbit

Fig. 3. The log-log plot of sample quality. The vertical axis represents MMD on different graph statistics, while the horizontal axis represents the size of sampled graphs.

Varying Graph Density. The four datasets mentioned in Section 4.1 are sparse graphs with an average connection of less than 6 (usually 2). To see what will happen when the graphs become denser, we experiment on BA graphs which can set the number of edges to attach from a new node to existing nodes (denoted as τ). While we have already tested on $\tau = 2$, we adopt $\tau \in \{10, 20\}$ to generate new datasets. We choose the best baseline BiGG and our model ForGen to be trained and tested on the denser graphs. Also, we report the ground truth BA model. Table 4 shows that even though denser graphs may contain more noises which can hinder learning-based methods, ForGen can still achieve the best results across three metrics.

4.4 Model Efficiency

Since BiGG is the most efficient deep autoregressive model and its lower time complexity has been proved, we compare ForGen with it in both training speed and inference speed. Table 5 reports the results of all five scale-free graph datasets we have. The first number of each item is the seconds model takes to finish a training iteration under the same batch size and the latter number is the inference time to generate 1k edges in a graph. Compared with BiGG's, the training speed of ForGen is faster when the graph is denser since more computation can be synchronized. When it comes to graph generation, ForGen can achieve about twice BiGG's speed on scale-free networks owing to its preferential forest design.

4.5 Ablation Study

Different Node Orderings. We choose four canonical node orderings mainly based on graph properties: the node degree descending ordering, BFS/DFS ordering (nodes with larger degree first), and the default ordering of the data.

Table 4. Quality test with increasing graph density.

Datasets	Methods	Degree	Clustering	Orbit
BA (τ=2)	BiGG	$1.42e^{-3}$	0.14	0.36
	ForGen (ours)	$1.25e^{-3}$	0.15	0.20
	Ground Truth	$1.13e^{-3}$	0.18	0.29
BA (τ=10)	BiGG	$7.45e^{-3}$	0.14	0.13
	ForGen (ours)	$5.50e^{-3}$	0.11	0.10
	Ground Truth	$4.72e^{-3}$	$8.07e^{-2}$	$6.31e^{-2}$
BA (τ=20)	BiGG	$1.48e^{-2}$	0.17	0.11
	ForGen (ours)	$1.40e^{-2}$	0.16	0.10
	Ground Truth	$1.54e^{-2}$	0.12	$6.25e^{-2}$

Table 5. Training/Inference speed on scale-free graphs.

	BA (τ=2)	BA (τ=10)	BA (τ=20)	Youtube	Amazon
BiGG	**0.79**/90.72	0.99/55.03	1.3/40.29	1.11/83.96	**0.66**/113.6
ForGen (ours)	1.07/**41.44**	**0.97/30.8**	**1.19/23.88**	**1.07/39.21**	0.76/**57.42**

Also, we test the combination of BFS and DFS. Table 6 shows the results of the Amazon dataset. We can see that both BFS and DFS can achieve good performance since they combine the topological information with preferential information (degree). However, node orderings lacking either information can hinder the learning process and the combination of different orderings with the same training strategy cannot help to improve the performance either. In summary, BFS or DFS ordering is recommended.

Table 6. Performance of models trained with different node orderings.

Node ordering	Degree	Clustering	Orbit
BFS	$1.08e^{-3}$	0.11	0.11
DFS	$1.01e^{-3}$	0.11	$9.57e^{-2}$
Degree descent	$1.51e^{-2}$	0.24	0.11
DFS+BFS	$2.08e^{-3}$	0.12	0.10
Default	0.23	0.35	0.11

Tree Size *b*. We next evaluate how the size of the first tree in the preferential forest (denoted as b) affects the quality. Since too small b will pay too much attention to noise and large b will lead to groups with fewer dependencies, we suggest $b \in \{5, 6, 7\}$. The experiment is conducted on the Youtube dataset. The results in Table 7 suggest that the selection of b within a reasonable range will not influence the effectiveness.

5 Conclusion

In this paper, we propose a forest-based graph generation model ForGen. First, we verify the wide existence of scale-free networks and skewed degree distri-

Table 7. Ablation study on tree size.

Tree Size	Degree	Clustering	Orbit
$b = 5$	$5.83e^{-3}$	0.11	0.10
$b = 6$	$8.06e^{-3}$	0.12	$9.90e^{-2}$
$b = 7$	$6.34e^{-3}$	0.12	0.11

butions on graphs of different sizes. To incorporate this important topological feature of real-world graphs, we design a multilevel data structure, which we call Preferential Forest, for the edge generation. We prove that the time complexity of generating an edge is $O(1)$ in the best case (which occurs frequently in scale-free networks) and $O(\log n)$ in the worst case. Together with a graph-level Fenwick tree, an autoregressive generation model that can generate graphs of any size is completed. By synchronization and an optimized tree traversal procedure, model scalability is further improved. We conduct extensive experiments on both synthetic and real graphs and prove ForGen's state-of-the-art sample quality from three different aspects. Furthermore, the generation speed of ForGen is two times faster than the most scalable model. In summary, the two-level tree-based model ForGen outperforms other autoregressive graph generators (like GraphRNN and BiGG) in both quality and efficiency.

References

1. Albert, R., Barabási, A.L.: Statistical mechanics of complex networks. Reviews of modern physics **74**(1), 47 (2002)
2. Barabási, A.: Network Science. Cambridge University Press (2016)
3. Barabási, A.L., Albert, R.: Emergence of scaling in random networks. science **286**(5439), 509–512 (1999)
4. Dai, H., Nazi, A., Li, Y., Dai, B., Schuurmans, D.: Scalable deep generative modeling for sparse graphs. In: International Conference on Machine Learning. pp. 2302–2312. PMLR (2020)
5. Erdos, P., Rényi, A., et al.: On the evolution of random graphs. Publ. Math. Inst. Hung. Acad. Sci **5**(1), 17–60 (1960)
6. Fenwick, P.M.: A new data structure for cumulative frequency tables. Software: Practice and experience **24**(3), 327–336 (1994)
7. Goyal, N., Jain, H.V., Ranu, S.: Graphgen: a scalable approach to domain-agnostic labeled graph generation. In: Proceedings of The Web Conference 2020. pp. 1253–1263 (2020)
8. Grover, A., Zweig, A., Ermon, S.: Graphite: Iterative generative modeling of graphs. In: International conference on machine learning. pp. 2434–2444. PMLR (2019)
9. Guo, X., Zhao, L.: A systematic survey on deep generative models for graph generation. arXiv preprint arXiv:2007.06686 (2020)
10. Huffman, D.A.: A method for the construction of minimum-redundancy codes. Proceedings of the IRE **40**(9), 1098–1101 (1952)
11. Jeong, H., Néda, Z., Barabási, A.L.: Measuring preferential attachment in evolving networks. EPL (Europhysics Letters) **61**(4), 567 (2003)

12. Kumar, R., Raghavan, P., Rajagopalan, S., Tomkins, A.: Extracting large-scale knowledge bases from the web. In: VLDB. vol. 99, pp. 639–650. Citeseer (1999)
13. Kunegis, J.: KONECT – The Koblenz Network Collection. In: Proc. Int. Conf. on World Wide Web Companion. pp. 1343–1350 (2013)
14. Kunegis, J., Blattner, M., Moser, C.: Preferential attachment in online networks: Measurement and explanations. In: Proceedings of the 5th Annual ACM Web Science Conference. pp. 205–214 (2013)
15. Leskovec, J., Chakrabarti, D., Kleinberg, J., Faloutsos, C., Ghahramani, Z.: Kronecker graphs: an approach to modeling networks. Journal of Machine Learning Research **11**(2) (2010)
16. Leskovec, J., Kleinberg, J., Faloutsos, C.: Graphs over time: densification laws, shrinking diameters and possible explanations. In: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining. pp. 177–187 (2005)
17. Li, Y., Zhang, L., Liu, Z.: Multi-objective de novo drug design with conditional graph generative model. Journal of cheminformatics **10**(1), 1–24 (2018)
18. Liao, R., Li, Y., Song, Y., Wang, S., Hamilton, W.L., Duvenaud, D., Urtasun, R., Zemel, R.: Efficient graph generation with graph recurrent attention networks. In: Proceedings of the 33rd International Conference on Neural Information Processing Systems. pp. 4255–4265 (2019)
19. Liu, C.C., Chan, H., Luk, K., Borealis, A.: Auto-regressive graph generation modeling with improved evaluation methods. In: Proc. NeurIPS Workshop Graph Represent. Learn. (2019)
20. Newman, M.E.: Power laws, pareto distributions and zipf's law. Contemporary physics **46**(5), 323–351 (2005)
21. Price, D.J.D.S.: Networks of scientific papers. Science pp. 510–515 (1965)
22. Simonovsky, M., Komodakis, N.: Graphvae: Towards generation of small graphs using variational autoencoders. In: International conference on artificial neural networks. pp. 412–422. Springer (2018)
23. Snijders, T.A., Nowicki, K.: Estimation and prediction for stochastic blockmodels for graphs with latent block structure. Journal of classification **14**(1), 75–100 (1997)
24. Tai, K.S., Socher, R., Manning, C.D.: Improved semantic representations from tree-structured long short-term memory networks. In: ACL (1) (2015)
25. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in neural information processing systems. pp. 5998–6008 (2017)
26. Watts, D.J., Strogatz, S.H.: Collective dynamics of 'small-world'networks. nature **393**(6684), 440–442 (1998)
27. Xiao, H., Huang, M., Zhu, X.: Transg: A generative model for knowledge graph embedding. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 2316–2325 (2016)
28. Xie, S., Kirillov, A., Girshick, R., He, K.: Exploring randomly wired neural networks for image recognition. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 1284–1293 (2019)
29. You, J., Liu, B., Ying, R., Pande, V., Leskovec, J.: Graph convolutional policy network for goal-directed molecular graph generation. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems. pp. 6412–6422 (2018)
30. You, J., Ying, R., Ren, X., Hamilton, W., Leskovec, J.: Graphrnn: Generating realistic graphs with deep auto-regressive models. In: International conference on machine learning. pp. 5708–5717. PMLR (2018)