Efficient Learning-based Community-Preserving Graph Generation

Sheng Xiang[1], Dawei Cheng[2*], Jianfu Zhang[3], Zhenwei Ma[4], Xiaoyang Wang[5], Ying Zhang[1]
[1]*AAII, University of Technology Sydney, Australia*
[2]*Dep. of Computer Science & Technology, Tongji University, China* [3]*Shanghai Jiao Tong University, China*
[4]*China Unionpay Co., Ltd., China* [5]*Zhejiang Gong Shang University, China*
[1]sheng.xiang@student.uts.edu.au;ying.zhang@uts.edu.au;
[2]dcheng@tongji.edu.cn; [3]c.sis@sjtu.edu.cn;
[4]mazhenwei@unionpay.com; [5]xiaoyangw@zjgsu.edu.cn;

***Abstract*—Graph generation is beneficial to comprehend the creation of meaningful structures of networks in a broad spectrum of applications such as social networks and biological networks. Recent studies tend to leverage deep learning techniques to learn the topology structures in graphs. However, we notice that the community structure, which is one of the most unique and prominent features of the graph, cannot be well captured by the existing graph generators. Moreover, the existing advanced deep learning-based graph generators are not efficient and scalable, which can only handle small graphs. In this paper, we propose a novel community-preserving generative adversarial network (CPGAN) for effective and efficient (scalable) graph simulation. We employ graph convolution networks in the encoder and share parameters in the generation process to transmit information about community structures and preserve the permutation-invariance in CPGAN. We conducted extensive experiments on benchmark datasets, including six sets of real-life graphs. The results demonstrate that CPGAN can achieve a good trade-off between efficiency (scalability) and graph simulation quality for real-life graph simulation compared with state-of-the-art baselines.**

I. Introduction

Graphs have been used to model relationships in a wide spectrum of applications such as social science, biology, and information technology [1], [2]. In some scenarios, the real-life graphs are not available due to a variety of reasons such as incomplete observability, privacy concern, and company/government policy. Thus, many graph techniques have been developed to simulate real-life graphs in various tasks such as modeling physical and social interactions and constructing knowledge graphs. For example, in financial fraud detection, generated graphs can be adopted to produce synthetic financial networks without divulging private information [3]. Moreover, graph generation techniques can also help us to better understand the distribution of graph structures and other features for the essential tasks. For instance, graph generators can be used to generate molecule [4] and formulas [5], which help to understand insights of the graph data.

Motivation. Due to its importance in both academia and industry, there is a long history of study on graph generators in many domains such as database, data mining, and machine learning (Please refer to [6] for a recent survey). Among these studies, the arguably most important line is the *general graph generator* which aims to learn a generative model to capture the structural distributions of the observed graphs regardless of the domains. Unless otherwise specified, the graph generators referred to in this paper are general graph generators. A large body of techniques have been developed in the literature which can be roughly divided into two categories: traditional graph generative models (e.g., [7]–[12]) and learning-based graph generative models (e.g., [13]–[15]). Generally speaking, the traditional approaches can efficiently generate large scale graphs based on some rules (e.g., [7]–[9]). However, the simulation quality of these models is not satisfactory for real-life graphs because they are hand-engineered to model some particular families of graphs and lack the capacity to learn a generative model directly from observed real-life graphs. With the advance of deep learning techniques, there is a clear trend in the literature where a variety of deep learning techniques such as recurrent neural network (RNN [13], [16]) and generative adversarial network (GAN [14], [17]) have been adopted to simulate real-life graphs. Though they have significantly improved the graph simulation quality by taking advantage of the sophisticated models, there are two major limitations.

(1) Community-preserving. Due to the complex nature of the graph, we cannot directly evaluate the goodness of the graph simulation by calculating similarity score between two graph distributions [1]. Thus, we have to resort to a variety of metrics each of which aims to quantitatively capture the likelihood of two graphs (graph distributions) from one perspective (e.g., degree distribution). We notice that the community structure, which is one of the most unique and prominent features of the graph, is neglected by most of the graph generators. Stochastic block models (SBM [20]), as well as its variants DCSBM [21], MMSB [10] and SBMGNN [15], BTER [22] and Chung-Lu model [23] consider the community structure. But there are only a few parameters in their models, which cannot properly capture the community structure of real-life graphs due to the simplicity of their generative models. It is well-known that the community structure preserves the inherent high-order structural property of the graph and hence

[*] Dawei Cheng is the corresponding author.

[1]Note that there are some similarity measures for two graphs such as graph edit distance [18] and maximum common subgraph [19]. But they cannot be applied to determine if two graphs are from the same distribution.

plays an important role in many downstream data analysis tasks such as link prediction and node classification. For instance, communities in a real-life network might represent real social groupings [24] and communities in a guarantee-loan network [25], [26] might represent dense loan relationships and financial institution groups (See Figure 1 as an example). This community information could help us to understand and exploit these networks more effectively [27]. Thus, in addition to the existing graph simulation quality evaluation metrics, we should also consider if a graph generator can well preserve the community structure of the observed real-life graph.

(2) Efficiency. Due to the complexity of the deep learning models, it is not a surprise that the emerging advanced deep learning-based general graph generative models are very time-consuming compared to the traditional models, and they can only handle small or medium-sized graphs in practice. For instance, GraphRNN [13] and GRAN [28] take RNN to generate the whole adjacency matrix, and NetGAN [14] uses random walks to assemble the whole graph, whose time complexity of training and inference procedures is $O(b \times n^2)$, where n is the number of graph nodes and b is the number of epochs. Considering that the efficiency and simulation quality are two important but contradictory requirements of graph generators in practice, it is desirable to develop a new deep learning model which can achieve a good trade-off between the efficiency (scalability) and the simulation quality.

Contribution. One may wonder if we can simply modify the existing deep learning-based generative graph generator models to preserve the community structure of the observed graphs. As to our best knowledge, this is non-trivial because it is challenging to integrate the community-preserving property in the learning and optimization of the graph generative models without sacrificing simulation quality. Thus, in this paper, we aim to design a new graph generative model which can better preserve the community structures of the observed graphs and have competitive performance on other evaluation metrics.

Recent advances in generative adversarial networks (GAN) have shown great successes in the graph simulation task (e.g., [14], [17], [29]). In this paper, we follow this line of research and propose a novel Community-Preserving Generative Adversarial Neural network (CPGAN) for the efficient and community-preserving graph generation. In particular, we propose a ladder network of the graph convolution and pooling layers as the permutation-invariant graph encoder. Before translating latent variables to a new graph, we leverage variational inference on the latent conjugate distributions to generate graphs naturally. The decoder (generator) is a full-connected network with a dot product to make link predictions once at all. Finally, the graph convolution and pooling layers are leveraged in an encoder (discriminator) to judge whether the community structure is well learned from the observed graphs.

In a nutshell, our principal contributions in this paper are summarized as follows:

- We propose a new graph generation model, Community-Preserving Generative Adversarial Network (CPGAN). It can not only preserve the community structure as well as other important properties of the real-life graphs but also reduce the graph simulation time and improve the scalability compared to other learning-based graph generation models.
- We carefully design the generator and discriminator in a unified GAN framework, in which the generator is in a hierarchical graph variational autoencoder that could learn permutation-invariant representations of input graphs and could generate new graphs from node representations (embeddings), and the discriminator to judge whether the embeddings are from real or simulated graphs.
- We introduce a differentiable ladder-shaped network to enable graph pooling and message transmitting in different community structure levels, which is more efficient and effective than simply stacking deeper graph convolution layers.
- We conduct extensive experiments on both synthetic and real-world graphs. Results show that our proposed model can achieve a good trade-off between graph simulation quality and efficiency (scalability) compared to the baseline methods.

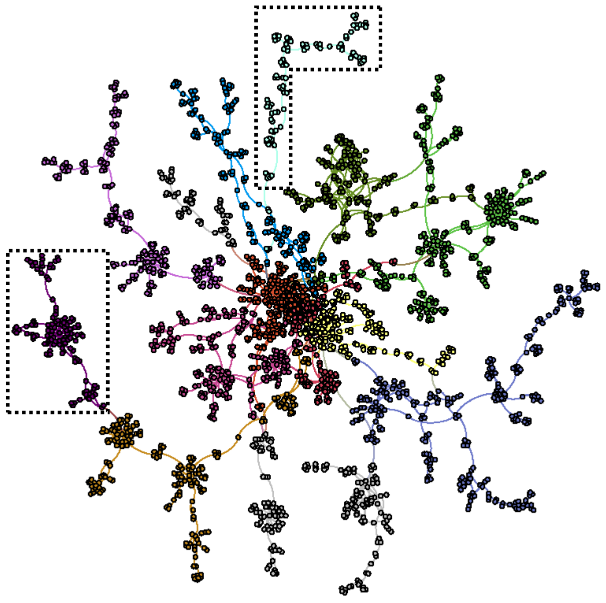


Fig. 1. An illustration of the communities of a real-life network.

II. Background

In this section, we first formally define the problem of graph generation in Section II-A, and then present the closely related works in Section II-B.

A. Problem Definition

We define a graph $G = (V, E)$ where V denotes a set of n nodes (vertices), and a set of m edges $E \subseteq V \times V$, where a tuple $e = (u, v) \in E$ represents an edge between two vertices u and v in V. The graph G can also be represented by an adjacency matrix $\mathbf{A} \in \{1, 0\}^{n \times n}$. Same as the literature, we assume G is an undirected graph, and hence the adjacency matrix of the graph is symmetric. Additionally, we denote the (optional) node-feature matrix associated with the graph as $\mathbf{X} \in \mathbb{R}^{n \times d}$ where n denotes the number of nodes and d

TABLE I
THE SUMMARY OF NOTATIONS

Notation	Definition
$\mathcal{G}$	the graph
A	adjacency matrix
X	node features
Z_{rec}	the node features reconstructed from input graph
$\mathcal{N}(\mu, \text{diag}(\sigma^2))$	the normal distributions
n	total number of vertices
m	total number of edges
$\mathcal{E}, \mathcal{D}$	the encoder and decoder
G, D	the generator and discriminator
$z^{(k)}$	the node features of the k-th level's community structure

denotes the dimension of the node feature. We summarize the notations in Table I.

Problem Statement. Given an observed graph G, a graph generative model aims to capture the structural distribution of the graph, such that a set of new graphs $\{G'\}$ with similar structural distribution can be generated.

Ideally, a general graph generative model should be able to generate new graphs which have exactly the same distribution as the observed graph. However, it is notoriously difficult to tell if two graphs are from the same distribution due to the complex nature of graph structure [13]. In practice, we have to resort to representative evaluating metrics in our experiments, each of which aims to quantitatively capture the likelihood of two graphs from one perspective (e.g., degree distribution). Please refer to section IV-A for more details. Below we introduce the evaluation metrics for the community-preserving which will be carefully considered by your graph generation model.

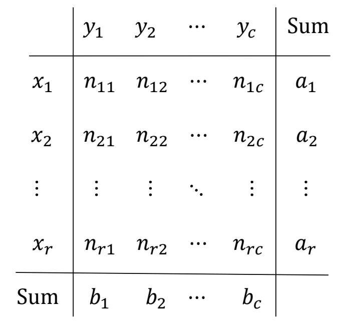
Evaluation of community-preserving. In this paper we aim to preserve the community structures of the training graphs. That is, we regard the community structures of the training graphs as the ground truth, and hopefully, the generated graph has the same community structure, where the goodness of the community-preserving is evaluated by two popular metrics: Adjusted Rand Index (namely **ARI**) and Normalized Mutual Information (namely **NMI**). Below are their detailed definitions.

Given a graph with n nodes and original community partition $Y_c = \{y_1, ..., y_c\}$, a community-preserving graph generative model generate a new graph with another community partition $X_r = \{x_1, ..., x_r\}$. Assume there is a bijective mapping between nodes of two graphs, we can formulate the similarity of two community partitions using *Rand Index* (namely RI) as follows:

$$\text{RI} = \frac{TP + TN}{TP + FP + FN + TN} \tag{1}$$

where TP is the number of true positives, i.e., the number of pairs of nodes that are in the same subsets in both X_r and Y_c, TN is the number of true negatives, i.e., the number of pairs of nodes that are in different subsets in X_r as well as Y_c. With the same rationale, FP and FN represent the number of false positives and the number of false negatives, respectively.

The Rand Index, however, suffers from one problem. For random data, it will be higher for low community counts than for high ones, because two nodes are more likely to be assigned together by chance. The ARI is a version of the RI corrected for the chance. According to Figure 2, a contingency table denotes the common nodes of two community partitions, with n_{ij} denoting the number of common nodes in the community x_i and y_j, $a_i = \sum_{j=1}^{c} n_{ij}$ and $b_j = \sum_{i=1}^{r} n_{ij}$. The ARI can be formulated in terms of the contingency table as follows:



	y_1	y_2	$\cdots$	y_c	Sum
x_1	n_{11}	n_{12}	$\cdots$	n_{1c}	a_1
x_2	n_{21}	n_{22}	$\cdots$	n_{2c}	a_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
x_r	n_{r1}	n_{r2}	$\cdots$	n_{rc}	a_r
Sum	b_1	b_2	$\cdots$	b_c	

Fig. 2. The contingency table of two community partitions X_r and Y_c.

$$\text{ARI} = \frac{\sum_{ij}\binom{n_{ij}}{2} - [\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}]/\binom{n}{2}}{\frac{1}{2}[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}] - [\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}]/\binom{n}{2}} \tag{2}$$

By calculating the ARI, we can quantitatively evaluate the similarity between the community structure of the generated graph and the observed graph. We can also use Mutual Information (MI) as evaluating metrics for community-preserving. MI is formulated as follows:

$$\text{MI} = \sum_{i=1}^{r}\sum_{j=1}^{c} \frac{n_{ij}}{N} \log \frac{N n_{ij}}{a_i b_j} \tag{3}$$

where N denotes the number of nodes. In practice, we use NMI, i.e., the normalized version of MI as our evaluating metric.

B. Related Work

In this subsection, we summarize the related works in two main areas: traditional graph generation methods (Section II-B1) and deep graph generative models (Section II-B2). As our model follows the line of the GAN-based graph generative model, some details of the algorithms in this category are also presented in Section II-B3.

1) Traditional Graph Generation Methods: The research on graph generative models has a long history [7], [8], [12]. Traditional approaches, such as B-A model [8], Chung-Lu model [23], Kronecker graphs [12], BTER [22], the exponential random graphs [30] and stochastic block models [10] etc., are carefully hand-engineered to model a particular family of graphs. For example, the exponential random graphs model (ERGMs) [30] relies on an expressive probabilistic model that learns weights over node features to model edge likelihoods, but in practice, this approach is limited by the fact that it only captures a set of graph-sufficient statistics. The Kronecker graph model [12] relies on Kronecker matrix products to efficiently generate large adjacency matrices. While scalable and able to learn some graph properties (e.g. degree distributions) from data, this approach remains highly constrained in terms of the graph structures that it can represent. The

BTER [22] model was proposed to correct the average clustering coefficient in each community, and correct the degree distribution through a two-level edge sampling process. BTER considers the community structure by explicitly modeling a graph as a two-level E-R graph. Note that SBM [20] and its variants DCSBM [21] and MMSB [10] also consider the community structure, but they are limited by the simplicity of the stochastic model, resulting in the poor performance in terms of community structure-preserving for real-life graphs. Specifically, only one parameter is used to capture each community (i.e., edges within this community), and one parameter is used to represent the connectivity probability for each pair of communities (i.e., edges between these two communities). A simple example about using SBM to generate new graphs with three communities is:

$$B = \begin{pmatrix} p_1 & 0 & 0 \\ 0 & p_2 & 0 \\ 0 & 0 & p_3 \end{pmatrix} \tag{4}$$

with this block matrix B, all edges will be generated within the communities and there is no edge across the communities. And the i-th community is equivalent to an E-R graph with edge probability p_i.

2) Deep Graph Generative Methods: In recent years, some techniques based on deep neural networks (e.g., VGAE [31], DeepGMG [32], GraphRNN [13], Graphite [33], GRAN [28], CondGen [17]) are proposed to tackle the problem of graph generation. They significantly improve the quality of graph generation compared to the traditional approaches. For instance, Graphite and VGAE use variational autoencoders (VAE) technique [31], [33] in which graph neural networks are applied for inference (encoding) and generation (decoding). As Graphite and VGAE assume a fixed set of vertices, they can only learn from a single graph. NetGAN performs more efficiently than VGAE by learning the graph's random walks, but it is not scalable because of the generation of the fixed size of graphs. In DeepGMG, graph neural networks are used to express probabilistic dependencies among nodes and edges of the graph, which can properly learn distributions over any arbitrary graph. However, it takes $O(mn^2D(G))$ operations to generate a graph with m edges, n vertices, and graph diameter $D(G)$, which also suffers from the scalability issue.

GraphRNN [13] generates a graph sequentially through recurrent neural networks (RNNs). But it is not permutation-invariant since computing the likelihood requires marginalizing out the possible permutations of the node orderings for the adjacency matrix. GRAN [28] improves the scalability of GraphRNN by generating one block of nodes and associated edges at each step in auto-regressive methods, which is still not permutation-invariant. CondGen [17] overcomes this permutation-invariance challenge by leveraging a GCN as the encoder and handling the graph generation problem in embedding spaces. Graph U-Nets [34] chooses specific nodes to realize upsampling and downsampling graphs to obtain graph representations. However, they do not consider the community structures of the observed graphs in the learning procedure. SBMGNN [15] is a variant of SBM equipped with deep learning techniques, but its graph neural networks are used to infer the parameters of the overlapping stochastic blockmodel, which is not directly relevant to the community-preserving property. Thus, there is no performance improvement in terms of community-preserving compared to other deep learning-based graph generative models.

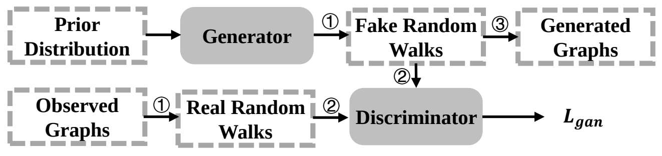


Fig. 3. A brief summary of NetGAN's architecture. NetGAN generates new graphs via three steps: (1) sampling random walks; (2) model training based on GAN framework; and (3) assembling the adjacency matrix.

3) GAN-based Graph Generator: Generative adversarial networks (GANs) [35] have shown remarkable results in various tasks such as image generation [36], image translation [37], super-resolution imaging [38], and multimedia synthesis [39]. GANs have also been utilized in network science tasks recently, such as network embedding [40], semi-supervised learning [41], and graph generation [13], [17]. For the graph generation task, prior structure knowledge specified by the sample dataset is crucial for graph generation, especially when preserving community structure. For the graph's community structure, some models using pooling strategy [34], [42] can be trained to represent communities (clusters) each time, but it is still challenging to represent and generate these community structures together. For instance, NetGAN generates graphs via random walks, which is nontrivial to preserve community structure. As to the time complexity of graph generation, generating a graph from NetGAN requires three steps according to Figure 3. The first and second steps require $O(kw)$ time complexity, where k denotes the number of walks, and w denotes the length of each walk. The third step requires $O(n^2)$ time. In practice, to well simulate the real-life graphs, steps 1 and 2 may require far more time than $O(n^2)$ due to the irreversible bias of random walks [43].

III. Our Approach

The key idea of CPGAN is that we can extract and reconstruct the community structure of a set of graphs by providing a community-preserving model regardless of the input permutation of each node. In this section, we outline the main challenges, show the framework and implementation details of our models, and introduce how to design the training process to reconstruct and generate new graphs.

A. Challenges

In this work, we hope to (1) learn the community structure of graphs using adjacency matrices A in the training set and reconstruct new graphs with a similar structure, and (2) generate new graphs using trained decoder $\mathcal{D}$ and samples from prior distributions $\mathcal{N}$. Below, we show four intrinsic challenges in community structure preserved graph generation, together with our unique contributions and the motivations of our solutions.

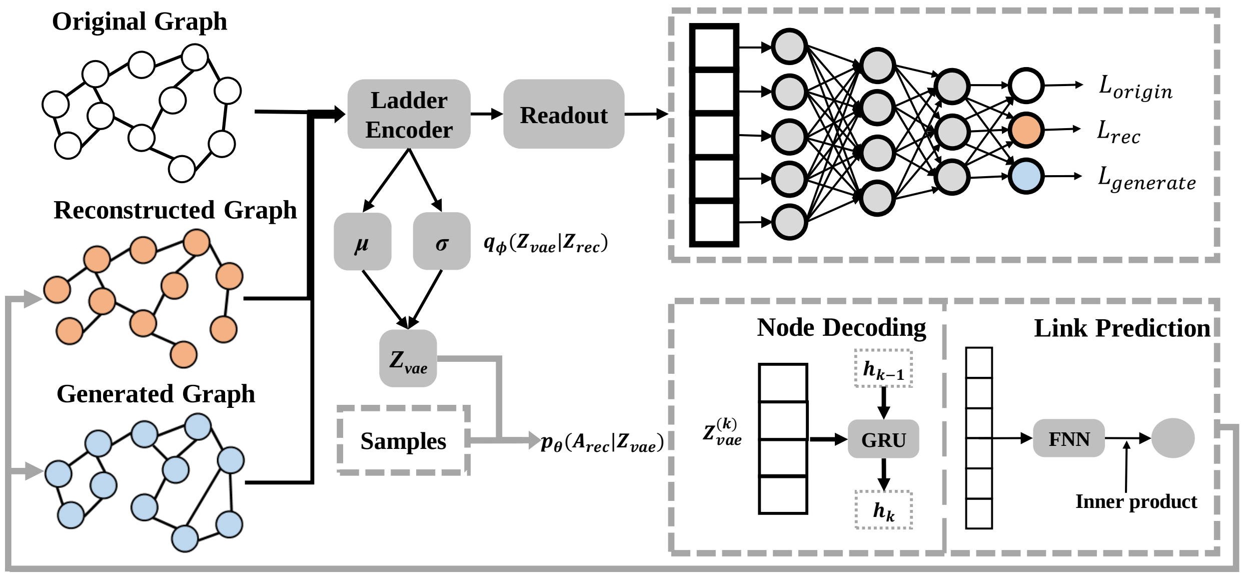


Fig. 4. The Framework of CPGAN

1) Community Structure Preserved Graph Generation: We notice that the community structure, which is one of the most unique and prominent features of the graph, is neglected by most of the graph generators. Inspired by hierarchical clustering, nodes with similar features can be clustered to the same class in a bottom-up fashion. On the other hand, in the representation learning of nodes, the clustering results can be used as important representation information about the community structure of nodes. Therefore, in practice, we introduce a hierarchical encoder to extract different levels of community structure as the input of the discriminator for the encoder of CPGAN, we need to use clustering results to enhance the graph discrimination level. For the decoding process of the generator, each node needs to exploit its community structure to enhance the generative performance. An intuitive solution is merging the node representations from different levels of community structure as the input of the decoder.

2) Permutation-Invariance: In our task, every graph with n nodes has $n!$ different permutations, which will result in an enormous number of adjacency matrices representing the same graph. Given a permutation matrix $\forall P \in \{0,1\}^{n\times n}$, we need the encoder and decoder to meet the requirements of the following equations to maintain permutation-invariance:

$$\begin{aligned}\textbf{Encoder: } & \mathcal{E}(PAP^T) = \mathcal{E}(A); \\ \textbf{Decoder: } & D(G(PZ)) = D(G(Z)).\end{aligned} \tag{5}$$

where Z denotes the samples from prior distributions. The adjacency matrices of different permutations may result in different graph representations if the model does not have permutation-invariance. Moreover, all permutation matrices need to be trained to learn the underlying representation of graphs. To address this issue, we implement our learning-based model through an all permutation-invariant architecture. Specifically, we ensure that all layers and objective functions are permutation-invariant to achieve this goal.

3) Scalability and Efficiency: The learning-based graph generative model can obtain better graph simulation quality. But there are some problems such as low efficiency and poor scalability, resulting in the incapability to generate real-life graphs. Therefore, it is desirable to develop a new deep learning model which can achieve a good trade-off between the efficiency (scalability) and the simulation quality. In practice, our can sample nodes without replacement to assemble subgraphs during training process to achieve a good trade-off between efficiency (scalability) and quality.

4) Differentiable Community Information Transmission: When extracting the community structure of a graph, Diffpool [42] can coarsen graphs layer by layer, similar to hierarchical clustering. And the model architecture of Diffpool for graph classification is consistent with that of our discriminator. Therefore, our model encodes graph representation through hierarchical clustering in a set of graphs and transmits this information to the decoder to generate graphs with a similar community structure. Moreover, our pooling layer is differentiable to achieve the objective function to preserve the community structure of observed graphs.

B. Model Architecture

Figure 4 illustrates the architecture of CPGAN. It includes the encoder ($\mathcal{E}$), the generator (G)/decoder ($\mathcal{D}$), and the discriminator (D). The upper right part is the discriminator, which gives its judgments on whether the graph is from real datasets. The lower right part is the generator, which decodes the community structures of a graph and reconstructs a new graph. For graph generation tasks, samples from prior distributions will be directly decoded to generate new graphs. And for graph reconstruction tasks, encoders in discriminator and generator share their parameters. Graphs reconstructed and generated will be fed into this model again to “fool” the discriminator. L_{origin}, L_{rec}, and $L_{generate}$ represent the loss of original graph, reconstructed graph, and generated graph, respectively.

As illustrated in Figure 4, for a graph $\mathcal{G}$, given its adjacency matrix A and feature matrix X, the structural information of the graph can be obtained by a ladder encoder. Regarding the community information, we assume that the observed graphs have ground truth community label Y_c, which can be otherwise obtained by applying existing community detection algorithms (e.g., [44]). The output of assignment matrix (will be introduced in Section III-C2) can be seen as the predicted node community assignments X_r. And the assignment matrix will be constrained by these ground truth labels. The community information and graph representation will be fed into the discriminator to determine whether the input graph is fake. At the same time, the coarsened graphs of each level will distribute their community structure features to original nodes through a differentiable hierarchical message transmission process. Then the series of community information of each node is decoded to enhance the reconstruction of the graph structure. The node representations sampled from the prior distribution can also be used to generate new graphs.

C. Ladder Message Transmission Encoder

Here we introduce a ladder-shaped encoder for this task so that our model can adaptively adjust the pooling strategy and extract the community structure information of nodes. We leverage node features X and adjacency matrix $A \in \{0,1\}^{n\times n}$ as the input of our proposed encoder. For each graph $\mathcal{G}$, we use identity matrix as its default node features X. Given node features $X \in \mathbb{R}^{n\times d}$ with d-dimension feature per node and adjacency matrix A, input graph $\mathcal{G}$ will be coarsened using stacked convolution and pooling layers.

1) Graph Convolution: As mentioned in Section II-B2, the classical message transmitting model is expressed by graph convolution networks (GCN) [17]. Post-transmitted message $Z \in \mathbb{R}^{n\times d'}$ can be calculated as follows:

$$Z = \sigma(GCN(X,A)) = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}XW) \quad (6)$$

where $\tilde{D} \in \mathbb{R}^{n\times n}$ denotes the degree matrix of $\tilde{A}$ with $\tilde{D}_{ii} = \sum_{j=0}^{n}\tilde{A}_{ij}$, $\tilde{A}$ denotes adjacency matrix containing self-loop with $\tilde{A} = A + I_n$, $W \in \mathbb{R}^{d\times d'}$ is trainable parameters in graph convolution layer with its kernel size d' and σ denotes activation function (default is the Rectified Linear Unit), and X denotes the node features derived from spectral embeddings of the adjacency matrix A with $X = X(A)$. Information can flow among nodes faster if we use some variants of $\tilde{A}$ (e.g. $\tilde{A} = A + A^2$) to improve the connectivity of graphs. The time complexity of graph convolution is $O(m+n)$, where m denotes the number of edges.

2) Graph Pooling: According to our experience, simply transmitting the message through GCNs can cause a very deep network to capture structure information, especially when we encounter large and sparse graphs with low connectivity. We need an effective way to obtain the hierarchical representations of a graph. Inspired by *Diffpool* [42], we can coarse a graph hierarchically and learn a strategy to coarse a set of graphs through a series of assignment matrices $S = \{S^{(l)} \in \mathbb{R}^{n_l\times n_{l+1}}, 1 \le l < k\}$, where n_l, n_{l+1}, and k denote the numbers of input nodes, output nodes, and layers respectively. The assignment matrix is calculated as follows:

$$\begin{aligned} Z^{(l)} &= \sigma(GCN_{l,embed}(X^{(l)}, A^{(l)})) \\ S^{(l)} &= softmax(GCN_{l,pool}(Z^{(l)}, A^{(l)})) \end{aligned} \quad (7)$$

where σ is Rectified Linear Unit activation function, $X^{(l)} \in \mathbb{R}^{n_l\times d_{l-1}}$ and $A^{(l)} \in \mathbb{R}^{n_l\times n_l}$ denote the feature matrix and adjacency matrix of n_l *cluster nodes*, respectively, $Z^{(l)} \in \mathbb{R}^{n_l\times d_l}$ denotes feature matrix with structure information of layer l, and two GCNs are leveraged to collect structure information and to infer the pooling strategy of layer l, respectively. Due to the multiple operations of graph convolution and pooling of one graph, we leverage a trick to use PairNorm [45] after each GCN to allow us to stack deep GCNs without over-smoothing. The assignment matrix can be seen as the predicted node community assignments. And the assignment matrix will be constrained by the ground truth labels (will be introduced in Section III-F2). Given assignment matrix $S^{(l)}$, the coarsened adjacency matrix $A^{(l+1)}$ and new embeddings $X^{(l+1)}$ can be generated as follows:

$$\begin{aligned} A^{(l+1)} &= {S^{(l)}}^T A^{(l)} S^{(l)} \\ X^{(l+1)} &= {S^{(l)}}^T Z^{(l)} \end{aligned} \quad (8)$$

Stacking graph convolution and pooling layers can obtain a series of node representations at different levels. In particular, if the layer k has just one node after pooling, the corresponding assignment matrix will be $\{1\}^{n_{k-1}}$, such that the graph pooling is equivalent to a graph readout `sum`. The overall time complexity of graph pooling is $O(m+n)$.

3) Graph Readout: Node representations of each graph are collapsed into a graph representation through graph readout. Therefore, the readout of output feature s_i of i-th level coarsened graph is calculated as follows:

$$\begin{aligned} s_i &= \frac{1}{n_i}\sum_{j=1}^{n_i} x_{ij} \\ s &= s_1 \oplus ... \oplus s_k \end{aligned} \quad (9)$$

where k is the number of layers for each graph, x_{ij} denotes the representation of j-th node of i-th level graph, and $\oplus ... \oplus$ denotes combining all representations in a new dimension. The time complexity of graph readout is $O(n)$. The final graph representation $s \in \mathbb{R}^{k\times d}$ is the input of the graph discriminator.

4) Graph Transposed Pooling: In order to reconstruct node representations, we need to properly depool a graph. Different from upsampling from a coarsened graph, we introduce a differentiable methodology to distribute information from the coarsened graph to a detailed graph. The proposed distributing method uses transposed versions of the similar assignment matrix. The transposed assignment matrix $S^{(l)}_{depool} \in \mathbb{R}^{n_{l+1}, n_l}$ is calculated as follows:

$$S^{(l)}_{depool} = softmax(GCN_{l,depool}(Z^{(l)}, A^{(l)})^T) \quad (10)$$

Therefore, reconstructed node representations $Z_{rec} \in \mathbb{R}^{n\times k\times d}$ are calculated as follows:

$$\begin{aligned} Z^{(l)}_{rec} &= \begin{cases} Z^{(l)} & (l=1) \\ \prod_{i=1}^{l-1} S^{(i)T}_{depool} \times Z^{(l)} & (l>1) \end{cases} \\ Z_{rec} &= Z^{(1)}_{rec} \oplus ... \oplus Z^{(k)}_{rec} \end{aligned} \quad (11)$$

where $\oplus ... \oplus$ denotes combining all node representations in a new dimension. After that, Z_{rec} is ready to be the input of our proposed decoder $\mathcal{D}$. The overall time complexity of transposed pooling is $O(m+n)$. Note that, in this work, we add a variational inference module to conjugate node latent distributions to control the output of our encoder.

D. Variational Inference

We leverage the variational inference before decoding node features for generating new graphs with observed hierarchical community structure distribution. We use $q_\phi(Z_{vae}|Z_{rec}) = \prod_{i=1}^{n} q_\phi(z_i|Z_{rec}).z_i \in Z_{vae}$ to achieve the mapping from the reconstructed features to the prior distributions $\mathcal{N}(\mu, \text{diag}(\sigma^2))$. And we choose a multi-layer perceptron (MLP) as our inference model. The inference process is formulated as follows:

$$\begin{aligned} g(Z_{rec}, \phi) &= \sigma(Z_{rec}\phi_0)\phi_1 \\ \bar{\mu} &= \frac{1}{n}\sum_{i=1}^{n} g_\mu(Z_{rec})_i \\ \bar{\sigma}^2 &= \frac{1}{n^2}\sum_{i=1}^{n} g_\sigma(Z_{rec})_i^2 \\ q_\phi(z_i|Z_{rec}) &\sim \mathcal{N}(\bar{z}|\bar{\mu}, \text{diag}(\bar{\sigma}^2)) \\ q_\phi(Z_{vae}|Z_{rec}) &= \sum_{i=1}^{n} q_\phi(z_i|Z_{rec}) \end{aligned} \tag{12}$$

where ϕ denotes the set of parameters in MLP, $g(\cdot)_i$ denotes the i-th row of $g(\cdot)$, and $Z_{vae} \in \mathbb{R}^{n\times k\times d'}$ is the output of variational inference module. The time complexity of inference module is $O(kn)$. According to [31], probabilistic variational reasoning can make the node representation far away from the zero-center, which intuitively makes the node representation more "sparse". We notice that this is helpful to preserve the node community structure.

After the variational inference module, we can select new node features from the prior distributions to generate new graphs. But we find that fully-connected networks alone cannot handle the task of generating graphs with complex and hierarchical community structure in section IV. So we proposed our graph decoder to address this issue.

E. Graph Decoder

Our proposed graph decoder consists of two steps: first decoding hierarchical graph representation sequences and then predicting node links. We embed hierarchical community structures with Gated Recurrent Unit (GRU) and obtain our node features h_k where k denotes the number of community structures. Decoded features are obtained by the following formula:

$$h_{l+1} = GRU(h_l, Z_{vae}^{(l+1)}). \quad (0 \le l < k) \tag{13}$$

where h_l denotes the hidden state of the coarsened graph, h_0 is a zero matrix, $Z_{vae}^{(l)} \in \mathbb{R}^{n\times d'}$ denotes the node features of the l-th coarsened graph, and h_k denotes the decoded node features with hierarchical community information. After obtaining the node representations, we give the link predictions as follows:

$$\begin{aligned} g_\theta(h_k) &= \sigma(h_k\theta_0)\theta_1 \\ p_\theta(A_{ij}|h_{k,i}, h_{k,j}) &= \sigma(g_\theta(h_{k,i})^T g_\theta(h_{k,j})) \\ p_\theta(A_{rec}|Z_{vae}) &= \prod_{i=1}^{n}\prod_{j=1}^{n} p_\theta(A_{ij}|h_{k,i}, h_{k,j}) \end{aligned} \tag{14}$$

where $g_\theta(h_{k,i})$ is a two-layer MLP to extract community information to help generate edges, $h_{k,i}$ denotes the feature of the i-th node, and $A_{rec} \in \mathbb{R}^{n\times n}$ denotes the probability matrix of link prediction. When training decoder on large graphs, to accelerate this process, we sample n_s ($n_s \ll n$) nodes to obtain $A_{rec} \in \mathbb{R}^{n_s\times n_s}$. Specifically, we sample nodes without replacement to assemble subgraphs with a strategy according to node degrees as follows: $P_i = \frac{deg_i}{\sum_{i=1}^{n} deg_i}$, where P_i is the probability to select node i, and deg_i denotes the degree of node i. Therefore, the time complexity of the graph decoder is $O(kn + n_s^2)$.

F. Discriminator and Optimization

1) Graph Discriminator: Discrimination task requires graph features obtained by the encoder, i.e., output matrix $s \in \mathbb{R}^{k\times d}$ of graph readout layer in section III-C3 with $s = \mathcal{E}(A)$. We leverage a two-layer MLP classifier as our discriminator D which is defined as

$$D_\phi(A) = \sigma(MLP(s, \phi)) \tag{15}$$

where ϕ denotes the parameters of MLP, and σ denotes the *sigmoid* activation function.

2) Discriminator Optimization: Formally, G and D play minimax game with value function $V(G, D)$ as follows:

$$\begin{aligned} \min_{\phi_G}\max_{\phi_D} V(D, G) = &\frac{1}{n}\sum_{i=1}^{n}\log(D(A_i)) \\ &+ \mathbb{E}_{p(Z_{vae})\sim q(\cdot|Z_{rec})}\log(1 - D(G(Z_{vae}))) \\ &+ \mathbb{E}_{p(Z_s)\sim \mathcal{N}(\cdot|\mathbf{0},\mathbf{I})}\log(1 - D(G(Z_s))) \end{aligned} \tag{16}$$

where Z_{vae} and Z_s are sampled from the approximate distributions and Gaussian prior distributions, respectively. Besides, to use the clustering results to enhance the level of discriminator, we introduce the clustering consistency $\mathcal{L}_{clus} = -\sum_{l=1}^{k}\text{logloss}(S^l, Y_c^l)$, where S^l denotes the assignment matrix introduced in Section III-C2, and Y_c^l denotes the ground-truth community partitions of observed graph. By default, we leverage *louvain* [44] community detection algorithm to obtain hierarchical community detection results as the ground-truth community partitions. In the training process, we need to update ϕ_D through ascending gradient by:

$$\nabla_{\phi_D} V(G, D) = \begin{cases} \nabla_{\phi_D}[\log(D(A)) + \mathcal{L}_{clus}] \\ \nabla_{\phi_D}\log(1 - D(G(Z))) \end{cases} \tag{17}$$

when judging graphs from real datasets, we update parameters using the upper part of equation 17. Note that, to ensure both community-structure preserving and other optimization objectives are well considered, our training process stops only when both $\mathcal{L}_{clus}$ and $\log(D(A))$ converge. When judging graphs generated, we update parameters using the lower part of equation 17.

3) Generator Optimization: The generator aims to minimize the log-probability that the discriminator correctly assigns to the graph reconstructed by G. Besides, to improve the performance of the decoder $\mathcal{D}$ and guarantee the permutation-invariance at the same time, we introduce the mapping consistency $\mathcal{L}_{rec}$ from CycleGAN [17] with

$\mathcal{L}_{rec} = \frac{1}{n}\sum_{i=1}^{n} ||\mathcal{E}(A_i) - \mathcal{E}(A'_i)||^2$, where A'_i denotes the fake adjacency matrix reconstructed from A_i. In practice, the collapse of encoder $\mathcal{E}$ can be controlled by the mapping consistency. We propose computing the gradient of the decoder with respect to $\phi_{\mathcal{D}}$ by descending gradient:

$$\begin{aligned} &\nabla_{\phi_{\mathcal{D}}}[V(G, D) - \mathcal{L}_{rec}(A', A)] \\ &= \nabla_{\phi_{\mathcal{D}}}[\mathbb{E}_{p(Z_{vae})\sim q(\cdot|Z_{rec})}\log(1 - D(G(Z_{vae}))) \\ &+ \mathbb{E}_{p(Z_s)\sim\mathcal{N}(\cdot|\mathbf{0},\mathbf{I})}\log(1 - D(G(Z_s))) \\ &- \frac{1}{n}\sum_{i=1}^{n} ||\mathcal{E}(A_i) - \mathcal{E}(A'_i)||^2] \end{aligned} \tag{18}$$

where A' denotes the reconstructed adjacency matrices. After updating the decoder, we propose computing the gradient of the encoder with respect to $\phi_{\mathcal{E}}$ by descending gradient:

$$\begin{aligned} &-\nabla_{\phi_{\mathcal{E}}}[\mathcal{L}_{prior}(q||p) + \mathcal{L}_{rec}(A', A)] \\ &= -\nabla_{\phi_{\mathcal{E}}}[D_{KL}(q(Z_{vae}|Z_{rec})||p(Z)) \\ &\quad + \frac{1}{n}\sum_{i=1}^{n} ||\mathcal{E}(A_i) - \mathcal{E}(A'_i)||^2] \end{aligned} \tag{19}$$

where the Gaussian prior $p(Z)$ is set with $p(Z) = \prod_{i=1}^{n} p(z_i) = \mathcal{N}(\bar{z}|\mathbf{0}, \mathbf{I})^n$, and $\mathcal{L}_{prior}(\cdot||\cdot)$ denotes calculating the Kullback-Leibler (KL) divergence between two distributions. With this modified encoder and decoder, the generation process can generate new graphs of arbitrary sizes and similar community structures.

G. Generating New Graphs

After the training, we sample n_s ($n_s \ll n$) nodes to obtain $A_{sub} \in \mathbb{R}^{n_s \times n_s}$ and assemble the output matrix $A_{out} \in \mathbb{R}^{n \times n}$ obtained from the generator and verified by the discriminator into a generated adjacency matrix. Specifically, we initialize an empty A_{out} and fill in edges genrated in each subgraph's adjacency matrix A_{sub} until the number of generated edge meets requirement. The binarization strategy that choosing a threshold to determine each edge and sampling strategy through Bernoulli distributions parameterized by A_{out} might lead to leaving out low-degree nodes and high variance output, respectively. To address these issues, we use the following strategy: (1) generating one edge for node i through sampling from category distribution parameterized by the i-th row of A_{out}; and (2) selecting top-k entries of A_{out} until the number of edges reaches a pre-defined number. The overall time complexity of generating new graphs is $O(n^2)$.

H. Discussion

Scalability. Though our proposed method is much more efficient and scalable compared to other learning-based approaches, it cannot compete with traditional approaches (e.g., BTER and ER) in terms of efficiency and scalability because the efficiency of the deep learning based approach relies on GPU which has limited memory size compared to that of CPU. Note that, in terms of graph training, CPGAN can handle large-scale graphs since we use a sampled graph in each training iteration. But CPGAN assumes the whole graph can be accommodated in the GPU memory in the graph simulation procedure, which limits the scalability of CPGAN. It will be interesting to develop more scalable learning-based solutions by making use of CPU memory or even hard disks, which is non-trivial because the swapping between GPU memory and other storage medium is time consuming.

Difference with existing learning-based methods. Same as other learning-based graph generators, some classical models such as VAE and CycleGAN are also used in our method. We focus on two new goals: efficiency (scalability) and community preserving for the learning-based model, which need the development of new techniques in this paper. For instance, we leverage VAE to separately infer the hierarchical structure information, which is helpful for community-preserving. Besides, the mapping consistency from CycleGAN and VAE are used for permutation-invariant graph generation, which is essential for our scalable implementation of our sampling strategy.

TABLE II
DETAILED STATS OF INCLUDED DATASETS

	#Nodes	#Edges	#Comm.	d_{mean}	CPL	GINI	PWE
Citeseer	3327	4732	473	2.8446	5.9389	0.6769	2.8757
PubMed	19717	44338	2488	4.4974	6.3369	0.8844	1.4743
PPI	2361	6646	371	5.8196	4.3762	0.7432	1.9029
3D Point Cloud	5037	10886	1577	4.3224	32.40	0.8278	1.9276
Facebook	50515	819090	8010	32.43	14.41	0.7164	1.5033
Google	875713	4322051	9863	9.871	6.3780	0.6729	1.8251

IV. EXPERIMENT

We perform extensive experiments to validate the effectiveness of our proposed methods. We describe the experimental settings first. Then, we give the experiment results of preserving community structure and generating realistic graphs compared with state-of-the-art baselines. At last, the model efficiency and memory consumption experiments are conducted, respectively.

A. Experiment Settings

Note that the source code and dataset used in the experiments are publicly available in GitHub (https://github.com/xiangsheng1325/CPGAN).

Dataset. We conduct experiments on six representative datasets in the literature including two citation networks (Citeseer and Pubmed) [46] [2], PPI [47] [3], 3D point cloud [48] [4], Facebook [49] [5], and Google web pages [50] [6]. These datasets span multiple domains and have different community structures. The graph statistics of these datasets are introduced on Table II, where "#Comm." denotes the number of communities. The detailed information of the six datasets are:

- **Citation Networks**. Citeseer and Pubmed are two typical citation networks, where nodes denote publications, and edges denote the citation relationships among publications. The Citeseer and Pubmed datasets contain 3327 and 19717 publications, and 4732 and 44338 citations, respectively.
- **PPI**. Protein-protein Interaction (PPI) network contains 2361 nodes and 6646 edges, each node representing one

[2]https://linqs.soe.ucsc.edu/data
[3]http://vlado.fmf.uni-lj.si/pub/networks/data/bio/Yeast/Yeast.htm
[4]http://www.first-mm.eu/data.html
[5]https://snap.stanford.edu/data/gemsec-Facebook.html
[6]https://snap.stanford.edu/data/web-Google.html

TABLE III
PERFORMANCE EVALUATION OF COMPARED MODELS FOR GRAPH COMMUNITY STRUCTURE PRESERVING TASKS IN EACH DATASET. NMI AND ARI MEASURE THE SIMILARITY BETWEEN COMMUNITY STRUCTURES OF GENERATED GRAPH AND THE ONE OF OBSERVED GRAPH, WHERE THE HIGHER IS BETTER.

Graph	Citeseer		Pubmed		PPI		3D Point Cloud		Facebook		Google	
	NMI(e-2)	ARI(e-2)	NMI(e-2)	ARI(e-2)	NMI(e-2)	ARI(e-2)	NMI(e-2)	ARI(e-2)	NMI(e-2)	ARI(e-2)	NMI(e-2)	ARI(e-2)
SBM	19.7$\pm$0.9	1.9$\pm$0.1	4.4$\pm$0.2	0.3$\pm$0.1	11.3$\pm$0.7	1.2$\pm$0.1	37.0$\pm$1.3	11.4$\pm$0.7	14.5$\pm$2.0	2.1$\pm$0.3	24.4$\pm$0.9	1.3$\pm$0.4
DCSBM	27.1$\pm$0.8	1.7$\pm$0.1	18.9$\pm$0.2	0.3$\pm$0.1	18.6$\pm$0.8	1.8$\pm$0.3	37.3$\pm$1.4	11.5$\pm$0.8	17.5$\pm$1.5	1.9$\pm$0.3	29.4$\pm$0.6	5.7$\pm$0.5
BTER	27.3$\pm$0.7	1.8$\pm$0.1	19.1$\pm$0.2	0.3$\pm$0.1	19.0$\pm$0.7	1.7$\pm$0.1	38.1$\pm$1.2	12.1$\pm$0.8	17.9$\pm$1.2	2.1$\pm$0.2	30.3$\pm$0.7	5.8$\pm$0.5
MMSB	26.7$\pm$0.9	4.4$\pm$1.0	OOM	OOM	15.4$\pm$0.6	0.8$\pm$0.4	7.1$\pm$0.4	1.3$\pm$0.3	OOM	OOM	OOM	OOM
VGAE	63.0$\pm$0.4	29.0$\pm$1.5	42.0$\pm$0.3	15.0$\pm$0.4	50.4$\pm$0.6	40.0$\pm$1.2	57.0$\pm$0.8	8.2$\pm$1.1	OOM	OOM	OOM	OOM
Graphite	62.8$\pm$0.7	28.2$\pm$2.1	43.0$\pm$0.5	15.1$\pm$0.4	52.3$\pm$0.8	33.4$\pm$1.9	58.8$\pm$0.4	13.2$\pm$0.3	OOM	OOM	OOM	OOM
SBMGNN	62.6$\pm$0.5	21.5$\pm$1.0	39.3$\pm$0.5	14.1$\pm$0.5	56.9$\pm$0.4	31.0$\pm$1.6	59.2$\pm$0.9	15.9$\pm$1.1	OOM	OOM	OOM	OOM
NetGAN	57.9$\pm$0.5	20.1$\pm$0.3	OOM	OOM	55.2$\pm$0.5	30.2$\pm$0.3	67.4$\pm$0.9	37.8$\pm$2.6	OOM	OOM	OOM	OOM
CPGAN	**72.5$\pm$0.4**	**44.3$\pm$1.5**	**45.8$\pm$0.9**	**34.1$\pm$1.1**	**57.0$\pm$0.7**	**44.2$\pm$1.3**	**70.6$\pm$0.6**	**39.9$\pm$1.4**	**54.7$\pm$1.0**	**28.4$\pm$1.6**	**38.7$\pm$0.5**	**30.8$\pm$0.5**

yeast protein. Edges are generated if there are interactions between two proteins.

- **3D Point Cloud**. Graph of points of household objects with 5037 nodes and 10886 edges, where nodes denote the objects, and edges are generated for k-nearest neighbors which are measured w.r.t Euclidean distance of the points in 3D space.
- **Facebook**. A real-world social network with 50515 nodes and 819090 edges, each node denoting one page. Edges are generated if there are mutual likes among them.
- **Google**. Web graph with 875713 nodes and 4322051 edges, where nodes represent web pages and edges represent hyperlinks between them.

Compared Methods. We compare our method with both the traditional models and recent deep graph generative models. All baseline models are designed to learn features on a set of graphs and generate new simulated graphs. The conventional baselines include: E-R [7], B-A [8], Chung-Lu [23], SBM [20], DCSBM [21], BTER [22], Kronecker [12], and MMSB [10]. The learning-based generative baselines are: VGAE [31], Graphite [33], SBMGNN [15], GraphRNN-S [13], NetGAN [14], and CondGen-R [17]. Note that we choose the scalable variant of GraphRNN and CondGen as baselines. In order to validate the effectiveness of CPGAN's sub-modules, we deploy some variants of our method, which are seperately denoted as CPGAN-C ("C" for "replacing the node decoding operation with a concatenation"), CPGAN-noV ("noV" for "not to use variational inference"), and CPGAN-noH ("noH" for "not to use hierarchical pooling") in the experiment. The CPGAN is our proposed graph generator in this paper.

Parameter Settings and Evaluation Metrics. The included algorithms and evaluating scripts are implemented and compiled through Python-3.6, PyTorch-1.8.1, CUDA-11.1, and GCC-4.8.5 in our experiments. The experiments are operated on a machine with Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz, 80 GB RAM and NVIDIA RTX 3090 with 24 GB memory. We use one CPU core and one GPU for every algorithm. In the experiment, we set the graph convolution kernel size to 128 in the ladder message transmission encoder. The learning rate is set to 0.001, the graph pooling size to 256. For various baselines, we employ the original hyperparameters settings of the compared models. To evaluate the performance of all methods, we leverage the following benchmark metrics in the experiment:

Deg.: Maximum Mean Discrepancy (MMD) of degree distribution measuring for the difference between degree distributions of two graphs.

Clus.: MMD of clustering coefficient distribution measuring for the difference between clustering coefficient distributions of two graphs.

CPL: The difference of characteristic path length between two graphs.

GINI: The difference of GINI index between two graphs, where GINI is a common measure for inequality in a degree distribution.

PWE: The difference of power-law exponent between two graphs.

NMI and ARI: To evaluate the community-preserving property, we compare the community structure similarity between the observed graphs and the generated graphs. Our evaluating method is based on the *louvain* [44] community detection algorithm. *louvain*, which has a complexity of $O(m+n)$, can quickly and hierarchically detect the community structure of the graph, and obtain the community membership of nodes unsupervised based on maximizing modularity Q. The modularity of community memberships of a graph is defined as follows:

$$Q = \frac{1}{2m}\sum_{i,j}[A_{ij} - \frac{d_i d_j}{2m}]\delta(c_i, c_j) \tag{20}$$

where m represents the number of edges, d_i denotes the degree of node i, the $\delta(u,v)$ is 0 when $u = v$ and 1 otherwise and c_i denotes the community membership to which node i is assigned. We suppose that graphs having the same community structure should have the same community detection results, so we use two popular clustering metrics[7], Normalized Mutual Information (NMI) and Adjusted Rand Index (ARI), to quantitatively evaluate the community structure of the generated graphs.

[7]https://scikit-learn.org/stable/modules/classes.html#clustering-metrics

TABLE IV
PERFORMANCE EVALUATION OF COMPARED MODELS FOR GRAPH GENERATION TASKS IN EACH DATASET. THE EVALUATION RESULTS ARE THE ABSOLUTE DIFFERENCES FROM TRUE MEASURES, WHERE THE LOWER IS BETTER.

Graph	Citeseer					3D Point Cloud					Google				
	Deg.	Clus.	CPL	GINI	PWE	Deg.	Clus.	CPL	GINI	PWE	Deg.	Clus.	CPL	GINI	PWE
E-R	1.27e-2	1.71e-2	17.5	8.86e-2	0.12	0.349	2	25.6	0.237	13.6	6.24e-2	1.36	13.17	3.99e-2	0.221
B-A	1.40e-2	1.25e-2	19.4	0.159	1.43	0.546	2	27.7	0.331	12.2	1.94e-2	1.36	11.1	6.16e-2	0.54
Chung-Lu	1.47e-2	1.73e-2	18.5	9.83e-2	0.15	0.353	2	25.7	0.222	13.7	6.48e-2	1.29	13.32	7.31e-2	0.624
SBM	1.36e-2	4.94e-3	**12.4**	7.87e-2	5.13e-2	0.317	1.99	23.4	0.209	13.8	0.111	0.886	6.93	0.113	0.892
DCSBM	2.40e-2	3.44e-3	13.3	0.142	8.14e-2	0.309	1.98	23.4	0.218	13.8	8.48e-2	0.865	11.8	9.17e-2	0.595
BTER	1.21e-2	2.71e-3	13.1	7.73e-2	**3.03e-2**	**0.301**	2	22.6	**0.207**	13.6	1.85e-2	0.834	6.67	3.93e-2	0.210
Kronecker	2.58e-2	1.91e-2	18.5	0.132	3.12e-2	0.370	2	26.8	0.240	13.8	0.102	1.28	15.1	5.19e-2	1.2
MMSB	2.98e-2	1.84e-2	17.9	0.173	0.186	0.339	2	25.9	0.234	13.7			OOM		
VGAE	0.123	3.78e-2	18.2	0.477	0.126	0.731	1.96	30	0.864	13.8			OOM		
GraphRNN-S	1.34e-3	**1.48e-3**	17.3	7.32e-2	0.176			OOM					OOM		
CondGen-R	8.42e-2	0.14	20.8	0.362	0.295	0.604	1.73	30.4	0.658	14.1			OOM		
NetGAN	**1.07e-3**	1.51e-3	16.5	0.136	0.154	0.415	1.72	26.3	0.542	14.6			OOM		
CPGAN	1.25e-3	2.26e-3	15.3	**7.23e-2**	9.32e-2	0.410	**1.49**	**18.1**	0.355	**10.8**	**1.47e-2**	**0.672**	**6.45**	**3.43e-2**	**0.118**

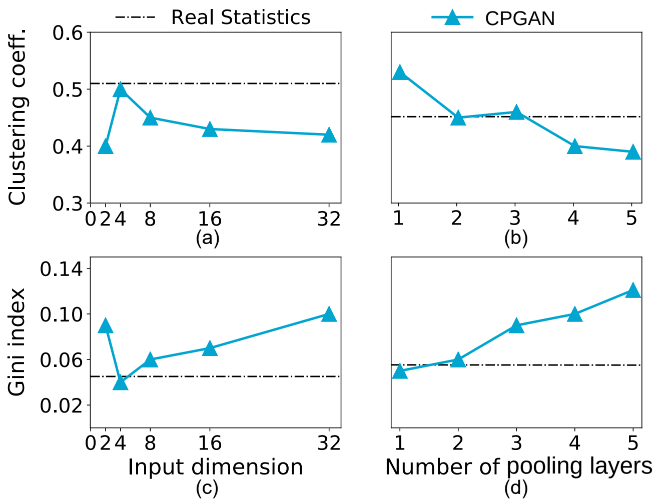


Fig. 5. Parameter sensitivity experiment results. Points closer to the real statistics are better.

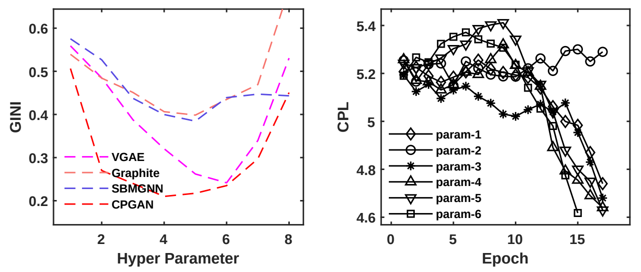


Fig. 6. Model robust experiment results. The *left* part is the performance of several compared methods, and the *right* part is the performance of different hyper parameter settings. Points lower are better.

B. Graph Generation

We introduce the experiments on several perspectives: preserving community structure, generative distribution distance, and parameter sensitivity. We conduct representative experiments and prove the superiority of our model in the task of graph generation, and some experiments with similar observations are excluded.

Preserving Community Structure. In this experiment, we first evaluate the generated graph by comparing community structures based on *louvain* [44] community detection algorithm. We compare the similarity of detection results between the observed graph and generated graph. Table III shows the performance of each method in preserving community structure, which is one of the main tasks of this paper. Note that several baseline methods are excluded because of the unstable node permutations. Some algorithms cannot do the graph simulation on some graphs due to the limit of memory, and the corresponding results are marked as "OOM" in the table.

The first 8 rows are the results of baseline methods. As expected, CPGAN demonstrates the best performance among all graph generators evaluated, especially for the evaluation of ARI. It is shown that BTER achieves the best performance compared with other traditional baselines. Nevertheless, its performance is not competitive compared to the learning-based models even most of them do not explicitly consider the community preserving property. As stressed in Section II-B2, SBMGNN does not utilize the deep neural network for the purpose of community preserving and hence does not show an advantage in the performance evaluation compared to other learning-based models.

Generative Distribution Distance. Table IV shows the performance of different methods in graph generation. Each evaluation metric represents the difference between real graphs and generated graphs. The first 12 rows are the results of baseline methods. It is shown that BTER has the best performance of the traditional graph generators (first 6 Lines). The deep learning-based generative models (Lines 7-12) can significantly improve the performance. It is shown that, in addition to the best performance on community preserving, CPGAN also demonstrates competitive performance on other quality measures on large graphs compared to the baselines. In two datasets with larger graph sizes, our method performs considerable improvements compared to the baselines. It is reported that CPGAN achieves one out of five best results in Citeseer, three out of five best results in 3D Point Cloud,

TABLE V
PERFORMANCE COMPARISON FOR GRAPH RECONSTRUCTION TASKS IN EACH DATASET.

Graph	PPI							Citeseer						
	Deg.	Clus.	CPL	GINI	PWE	Train NLL	Test NLL	Deg.	Clus.	CPL	GINI	PWE	Train NLL	Test NLL
VGAE	0.257	1.69	**6.11**	0.342	0.633	1.96	3.61	9.01e-2	1.6	1.45	0.263	0.149	2.26	3.78
Graphite	0.315	0.815	10.9	0.362	0.760	2.09	4.38	0.306	1.53	2.14	0.311	1.17	2.41	4.15
SBMGNN	0.356	1.61	10.9	0.397	0.777	2.20	4.00	0.217	1.32	2.14	0.358	0.517	2.31	4.26
CondGen	0.139	1.16	12.8	0.231	1.09	2.07	3.82	0.166	1.13	3.57	0.196	1.54	2.47	3.97
CPGAN	**6.21e-2**	**0.243**	11.31	**7.43e-2**	**0.437**	**1.84**	**3.52**	**8.49e-2**	**0.498**	**1.35**	**1.38e-2**	**3.16e-2**	**1.78**	**3.68**

TABLE VI
PERFORMANCE EVALUATION OF SUB-MODELS FOR GRAPH COMMUNITY PRESERVING AND GRAPH GENERATION TASKS IN 3 DATASETS. THE NMI AND ARI ROWS SHOW THE COMMUNITY-PRESERVING MEASURES OF GENERATED GRAPHS, WHERE THE HIGHER IS BETTER, WHILE OTHERS ARE THE ABSOLUTE DIFFERENCES FROM TRUE MEASURES, WHERE THE LOWER IS BETTER.

Graph	PubMed				PPI				Facebook			
	NMI(e-2)	ARI(e-2)	Deg.	Clus.	NMI(e-2)	ARI(e-2)	Deg.	Clus.	NMI(e-2)	ARI(e-2)	Deg.	Clus.
CPGAN-C	32.1	14.5	2.38e-3	2.23e-3	51.2	39.3	2.47e-3	1.35e-2	53.3	26.1	1.20e-3	1.43e-2
CPGAN-noV	31.3	14.3	3.03e-3	5.14e-3	50.5	39.0	2.77e-3	1.76e-2	52.9	25.3	1.24e-3	1.56e-2
CPGAN-noH	28.8	13.2	3.96e-3	6.52e-3	49.7	38.4	3.49e-3	2.30e-2	50.1	23.2	1.96e-3	1.79e-2
CPGAN	**45.8**	**34.1**	**2.08e-3**	**1.81e-3**	**57.0**	**44.2**	**2.35e-3**	**1.12e-2**	**54.7**	**28.4**	**1.18e-3**	**1.35e-2**

and all five best results in the Google web graph dataset. According to the experiments, CPGAN achieves the best results on PubMED and FaceBook datasets, similar to the conclusions on Google datasets. And CPGAN shows the best performance on 3D Point Cloud data sets. Besides, CPGAN has competitive performance on PPI dataset, similar to the conclusions on 3D Point Cloud data sets. We notice that on PubMed, FaceBook, Google datasets, the compared learning-based baseline algorithms all lead to memory overflow due to their high space complexity. As expected, CPGAN always outperforms CPGAN-C, demonstrating the effectiveness of our new graph decoding module.

Parameters Sensitivity. Figure 5 illustrates the result of parameter sensitivity experiments. We report the statistical differences between the generated graph and the real graph according to the changes of spectral embedding dimension (Figures 5a and 5c) and the number of hierarchies ((Figures 5b and 5d)), where the number of hierarchies denotes the number of node clustering results in ladder encoder. It is clear the level of hierarchies around two achieves the best performance, which proves the essence of preserving community information in the learning process. Besides, the change of the dimension has no significant influence on the performance of the model. Based on the sensitivity experiment, we chose the best parameter (the input dimension of four and the level of hierarchical structures of two) in our experiment.

The left part of Figure 6 illustrates the comparison of model robustness and the training difficulty of our methods. We select the models with similar architecture and hyper-parameters to compare. When the hyper-parameters are traversed in the interval, our method is obviously more robust than other methods. Other experimental results on model robustness can draw the same conclusion. Based on the model robustness comparison experiment, we chose the best parameter settings of other baseline methods (the input dimensions and the hidden dimensions). The right part of Figure 6 illustrates the tuning difficulty of hyper-parameters. Our method is stable and has few collapses or instability in the case of different hyper-parameters. Based on the model tuning difficulty experiment, we chose the best training strategy (the learning rate of 0.001 and decay of 0.3 per 400 epochs) in our experiment.

TABLE VII
TIME CONSUMPTION (SECONDS) PER GRAPH GENERATION.

#Nodes	0.1k	1k	10k	100k
E-R	$\mathbf{4.6}e^{-4}$	$9.0e^{-3}$	0.46	10.1
B-A	$1.0e^{-3}$	$1.2e^{-2}$	**0.11**	1.17
Chung-Lu	$7.2e^{-4}$	$2.5e^{-3}$	0.18	2.38
SBM	$6.1e^{-3}$	0.09	2.58	37.1
DCSBM	$6.2e^{-3}$	0.09	2.69	39.3
BTER	$1.28e^{-3}$	$\mathbf{1.9}e^{-3}$	0.16	**0.25**
MMSB	$6.1e^{-3}$	0.09	2.56	-
Kronecker	$8.5e^{-3}$	0.08	1.00	9.69
GraphRNN-S	0.27	4.74	63.6	-
VGAE	$4.2e^{-3}$	0.04	0.38	-
Graphite	$6.1e^{-3}$	0.06	0.64	-
SBMGNN	0.01	0.11	1.18	-
NetGAN	$8.7e^{-3}$	0.09	1.12	-
CondGEN-R	$8.3e^{-3}$	0.15	-	-
CPGAN	$9.1e^{-3}$	0.08	0.95	86.1

TABLE VIII
TIME CONSUMPTION (MINUTES) OF THE ENTIRE TRAINING PROCESS.

#Nodes	0.1k	1k	10k	100k
MMSB	0.11	0.91	40.3	-
Kronecker	1.39	1.55	**3.25**	**4.73**
GraphRNN-S	1.63	15.4	161	-
VGAE	**0.06**	**0.42**	9.75	-
Graphite	0.07	0.47	10.6	-
SBMGNN	0.08	0.63	12.4	-
NetGAN	0.27	2.80	31.1	-
CondGEN-R	0.18	25.3	-	-
CPGAN	0.35	0.70	6.39	32.9

C. *Graph Reconstruction*

In this experiment, we use the complete dataset of PPI and Citeseer. We randomly select 80% edges of the dataset

TABLE IX
PEAK GPU MEMORY USAGE (MIB) DURING TRAINING

#Nodes	0.1k	1k	10k	100k
MMSB	**1575**	**1709**	18529	OOM
GraphRNN-S	1913	1959	5501	OOM
VGAE	1719	1759	4799	OOM
Graphite	1719	1761	4819	OOM
SBMGNN	1719	1767	5243	OOM
NetGAN	2237	2552	5008	OOM
CondGEN-R	1722	1789	-	-
CPGAN	1728	1760	**2467**	**7930**

as the training set and employ the model to reconstruct the whole graph, including the rest 20% test set edges. We compute the negative log-likelihood (NLL) of the score given by the discriminator and report the average number from train and test data sets. We exclude the E-R, B-A, and other methods that cannot employ to reconstruct graphs, and the experiments on other dataset preserves the conclusion that our method improves the performance among other baselines. Table V reports the experimental results. As we can see, our method achieves the most competitive results in the PPI dataset and perform the best in the Citeseer dataset with significant improvements to VGAE, Graphite, SBMGNN, and CondGen. The results are in accord with graph generation experiments, which prove the effectiveness of our method in realistic graph generation. We notice that the performance of the GAN-based models, including CPGAN, are not good for the measure of CPL on graphs with low CPL value (e.g., the CPL value of PPI dataset is 4.38 as shown in Table II, which is the smallest one in 6 datasets). The reason is that the edges of graphs with low CPL value tends to be denser than ones with high CPL value. The dense edges may lead to a more complex discriminator. So the generator part of GAN will make more complex decisions to avoid the strong attack from discriminator part of GAN. Then the structure of dense graph generated by the GAN-based graph generative model is not stable in adversarial training process. And the CPL is a structure-sensitive measurement of generated graphs. Therefore, GAN-based models (e.g., CondGen and CPGAN) perform worse than other baseline models on graphs with small CPL value.

D. Ablation Study

We evaluate the effectiveness of each sub-module in our proposed method in this subsection. In Table VI, the first 3 rows show the performance of our proposed model's variations. It can be seen that our proposed method outperforms all the variants. The row 2 of Table VI shows that the variant CPGAN-noV performs worse than our proposed CPGAN, which demonstrates the effectiveness of variantional inference. We can also find that the variant CPGAN-noH shows the worst performance among these variants, which demonstrates the effectiveness of our proposed model's components, especially for the ladder encoder with hierarchical pooling.

E. Model Efficiency and Scalability

We evaluate the efficiency and scalability of the graph generators in this subsection. Table VII reports the time consumption of inferring a new graph where the number of nodes varies from 0.1K to 100K. Table VIII chronicles the time consumption of the entire training process and Table IX details the peak memory usage during the training process. As expected, the traditional graph generators such as BTER, Chung-Lu, E-R, B-A, SBM, DCSBM, and Kronecker outperform the learning-based graph generators in terms of efficiency and scalability, especially on large graphs. On the other hand, it is shown that CPGAN has the best efficiency and scalability among the learning-based approaches when the size of the graph grows. For instance, only CPGAN can handle the graph with size 100K in Tables VII, VIII and IX.

F. Summary

It is reported that traditional graph generators (e.g., BTER and ER) significantly outperform learning-based graph generators in terms of efficiency and scalability, especially on large graphs. Due to the limitation of the learning-based methods as discussed in subsection III-H, all learning-based methods, including CPGAN, cannot handle larger dataset with millions-scale nodes under current experiment setting (e.g., 24GB GPU memory). This suggests that BTER is the best choice when simulating large-scale graphs since it has the best graph simulation quality among all traditional graph generators and competitive performance in terms of efficiency and scalability. Nevertheless, traditional graph generators are not good choices in some applications where high simulation quality is required on real-life graphs. On the other hand, existing learning-based approaches can achieve good simulation quality, but have poor performance in terms of efficiency and scalability. It is shown that CPGAN has the best efficiency and scalability among the learning-based approaches when the size of the graph grows. Given the best performance on community-preserving and competitive performance on other simulation quality evaluation metrics, *we boast that CPGAN achieves a very good trade-off between graph simulation quality and efficiency (scalability) compared to other approaches.*

V. CONCLUSION

In this paper, we propose a deep generative model named CPGAN to simulate real-life graphs. Our model is designed to keep community structure together with other important properties in the graph simulation process. The simulation quality and efficiency (scalability) are two important but contradictory requirements of graph generators in practice, and our method can achieve a good trade-off, especially in the large real-world networks, compared to existing general graph generators.

ACKNOWLEDGEMENTS

This work was supported by the National Science Fundation of China (62102287). Ying Zhang is supported by FT170100128 and ARC DP210101393. Xiaoyang Wang is supported by ZJNSF LQ20F020007 and ZJNSF LY21F020012. The authors want to thank Chenhao Xu for the useful discussion on this paper.

References

[1] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2016, pp. 855–864.

[2] C. Zang, P. Cui, C. Faloutsos, and W. Zhu, "Long short memory process: Modeling growth dynamics of microscopic social connectivity," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 565–574.

[3] E. M. Fich and A. Shivdasani, "Financial fraud, director reputation, and shareholder wealth," *Journal of Financial Economics*, vol. 86, no. 2, pp. 306–336, 2007.

[4] J. You, B. Liu, R. Ying, V. Pande, and J. Leskovec, "Graph convolutional policy network for goal-directed molecular graph generation," in *NeurIPS*. Curran Associates Inc., 2018, p. 6412–6422.

[5] J. You, H. Wu, C. Barrett, R. Ramanujan, and J. Leskovec, "G2sat: Learning to generate sat formulas," *NeurIPS*, 2019.

[6] A. Bonifati, I. Holubová, A. Prat-Pérez, and S. Sakr, "Graph generators: State of the art and open challenges," *ACM Comput. Surv.*, 2020.

[7] P. Erdős and A. Rényi, "On random graphs i. publicationes mathematicae (debrecen)," 1959.

[8] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Reviews of modern physics*, vol. 74, no. 1, p. 47, 2002.

[9] D. Watts and S. Strogatz, "Collective dynamics of 'small-world' networks (see comments)," *Nature*, pp. P.440–442, 1998.

[10] E. M. Airoldi, D. M. Blei, S. E. Fienberg, and E. P. Xing, "Mixed membership stochastic blockmodels," *Journal of machine learning research*, vol. 9, no. Sep, pp. 1981–2014, 2008.

[11] L. Akoglu and C. Faloutsos, "RTG: a recursive realistic graph generator using random typing," *Data Min. Knowl. Discov.*, pp. 194–209, 2009.

[12] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, "Kronecker graphs: An approach to modeling networks," *Journal of Machine Learning Research*, vol. 11, no. Feb, pp. 985–1042, 2010.

[13] J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec, "Graphrnn: Generating realistic graphs with deep auto-regressive models," in *International Conference on Machine Learning*, 2018, pp. 5694–5703.

[14] A. Bojchevski, O. Shchur, D. Zügner, and S. Günnemann, "Netgan: Generating graphs via random walks," in *International Conference on Machine Learning*, 2018, pp. 610–619.

[15] N. Mehta, L. Carin, and P. Rai, "Stochastic blockmodels meet graph neural networks," in *ICML*, 2019, pp. 4466–4474.

[16] D. Cheng, Z. Niu, and L. Zhang, "Delinquent events prediction in temporal networked-guarantee loans," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[17] C. Yang, P. Zhuang, W. Shi, A. Luu, and P. Li, "Conditional structure generation through graph variational generative adversarial nets," in *Advances in neural information processing systems*, 2019.

[18] A. Sanfeliu and K.-S. Fu, "A distance measure between attributed relational graphs for pattern recognition," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, pp. 353–362, 1983.

[19] V. Kann, "On the approximability of the maximum common subgraph problem," in *STACS*, 1992.

[20] P. W. Holland, K. B. Laskey, and S. Leinhardt, "Stochastic blockmodels: First steps," *Social Networks*, vol. 5, no. 2, pp. 109–137, 1983.

[21] B. Karrer and M. E. J. Newman, "Stochastic blockmodels and community structure in networks," *Physical Review E*, vol. 83, no. 1 Pt 2, p. 016107, 2011.

[22] T. G. Kolda, A. Pinar, T. Plantenga, and S. Comandur, "A scalable generative graph model with community structure," *SIAM J. Sci. Comput.*, vol. 36, 2014.

[23] F. R. K. Chung and L. Lu, "The average distances in random graphs with given expected degrees," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99, pp. 15 879 – 15 882, 2002.

[24] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *Proc Natl Acad U S A*, vol. 99, no. 12, pp. 7821–7826, 2002.

[25] D. Cheng, Z. Niu, Y. Tu, and L. Zhang, "Prediction defaults for networked-guarantee loans," in *2018 24th International Conference on Pattern Recognition (ICPR)*. IEEE, 2018, pp. 361–366.

[26] Z. Niu, R. Li, J. Wu, D. Cheng, and J. Zhang, "iconviz: Interactive visual exploration of the default contagion risk of networked-guarantee loans," in *2020 IEEE conference on visual analytics science and technology (VAST)*. IEEE, 2020, pp. 84–94.

[27] D. Cheng, C. Chen, X. Wang, and S. Xiang, "Efficient top-k vulnerable nodes detection in uncertain graphs," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[28] R. Liao, Y. Li, Y. Song, S. Wang, W. Hamilton, D. K. Duvenaud, R. Urtasun, and R. Zemel, "Efficient graph generation with graph recurrent attention networks," in *Advances in Neural Information Processing Systems*, 2019, pp. 4255–4265.

[29] S. Xiang, D. Wen, D. Cheng, Y. Zhang, L. Qin, Z. Qian, and X. Lin, "General graph generators: experiments, analyses, and improvements," *The VLDB Journal*, pp. 1–29, 2021.

[30] S. Wasserman and P. Pattison, "Logit models and logistic regressions for social networks: I. an introduction to markov graphs andp," *Psychometrika*, vol. 61, no. 3, pp. 401–425, 1996.

[31] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *NIPS Workshop on Bayesian Deep Learning*, 2016.

[32] D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling, "Semi-supervised learning with deep generative models," in *Advances in neural information processing systems*, 2014, pp. 3581–3589.

[33] A. Grover, A. Zweig, and S. Ermon, "Graphite: Iterative generative modeling of graphs," in *International Conference on Machine Learning*, 2019, pp. 2434–2444.

[34] H. Gao and S. Ji, "Graph u-nets," in *Proceedings of the 36th International Conference on Machine Learning*, 2019.

[35] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[36] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *International conference on machine learning*, 2017, pp. 214–223.

[37] Y. Choi, M. Choi, M. Kim, J.-W. Ha, S. Kim, and J. Choo, "Stargan: Unified generative adversarial networks for multi-domain image-to-image translation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8789–8797.

[38] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation," in *International Conference on Learning Representations*, 2018.

[39] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4401–4410.

[40] H. Gao, J. Pei, and H. Huang, "Progan: Network embedding via proximity generative adversarial network," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2019, pp. 1308–1316.

[41] M. Ding, J. Tang, and J. Zhang, "Semi-supervised learning on graphs with generative adversarial nets," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. ACM, 2018, pp. 913–922.

[42] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Advances in neural information processing systems*, 2018, pp. 4800–4810.

[43] L. Rendsburg, H. Heidrich, and U. von Luxburg, "Netgan without gan: From random walks to low-rank approximations," in *ICML*, 2020.

[44] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, oct 2008. [Online]. Available: https://doi.org/10.1088/1742-5468/2008/10/p10008

[45] L. Zhao and L. Akoglu, "Pairnorm: Tackling oversmoothing in {gnn}s," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=rkecl1rtwB

[46] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad, "Collective classification in network data," *AI Mag.*, vol. 29, no. 3, pp. 93–106, 2008. [Online]. Available: https://doi.org/10.1609/aimag.v29i3.2157

[47] D. Bu, Y. Zhao, L. Cai, H. Xue, X. Zhu, H. Lu, J. Zhang, S. Sun, L. Ling, N. Zhang, G. Li, and R. Chen, "Topological structure analysis of the protein-protein interaction network in budding yeast," *Nucleic acids research*, pp. 2443–2450, 2003.

[48] m. neumann, p. moreno, l. antanas, r. garnett, and k. kersting, "Graph kernels for object category prediction in task-dependent robot grasping," 2013.

[49] B. Rozemberczki, R. Davies, R. Sarkar, and C. Sutton, "Gemsec: Graph embedding with self clustering," in *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2019*. ACM, 2019, pp. 65–72.

[50] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters," *Internet Math.*, pp. 29–123, 2009.