Scalable Learning-based Community-Preserving Graph Generation

Sheng Xiang, Chenhao Xu, Dawei Cheng and Ying Zhang

***Abstract*—Graph generation plays an essential role in understanding the formation of complex network structures across various fields, such as biological and social networks. Recent studies have shifted towards employing deep learning methods to grasp the topology of graphs. Yet, most current graph generators fail to adequately capture the community structure, which stands out as a critical and distinctive aspect of graphs. Additionally, these generators are generally limited to smaller graphs because of their inefficiencies and scaling challenges. This paper introduces the Community-Preserving Graph Adversarial Network (CPGAN), designed to effectively simulate graphs. CPGAN leverages graph convolution networks within its encoder and maintains shared parameters during generation to encapsulate community structure data and ensure permutation invariance. We also present the Scalable Community-Preserving Graph Attention Network (SCPGAN), aimed at enhancing the scalability of our model. SCPGAN considerably cuts down on inference and training durations, as well as GPU memory usage, through the use of an ego-graph sampling approach and a short-pipeline autoencoder framework. Tests conducted on six real-world graph datasets reveal that CPGAN manages a beneficial balance between efficiency and simulation quality when compared to leading-edge baselines. Moreover, SCPGAN marks substantial strides in model efficiency and scalability, successfully increasing the size of generated graphs to the 10 million node level while maintaining competitive quality, on par with other advanced learning models.**

***Index Terms*—Graph Generation, Generative Adversarial Network, Graph Neural Network.**

I. Introduction

GRAPHS are widely used in relation modeling in various fields like social science, information technology, and biology [1], [2]. However, it is difficult to obtain real-life graphs in some scenarios due to privacy issues, incomplete observability, policy restrictions, etc. Therefore, researchers have developed numerous graph techniques to simulate real-life graphs in many tasks such as building knowledge graphs and modeling interactions. For example, synthetic financial networks can be generated without revealing sensitive information in financial fraud detection [3]. In addition, graph generation techniques contribute to a better understanding of graph structure distributions and other features. For example, graph generators aid in generating molecules [4], materials [5] and formulass [6], providing valuable insights into graph data.

Motivation. The study of graph generators has a rich history across various fields like databases, data mining, and machine learning [7]. Among the most important of these researches is *general graph generator*, which learns a generative model capturing graph structure distributions in spite of domains. Unless otherwise stated, the graph generators mean general graph generators in this paper. In this field, techniques have evolved into two main streams: traditional graph generators [8]–[13] and learning-based graph generators [14]–[16]. Traditional approaches produce large-scale graphs efficiently though predefined rules [8]–[10], yet they fall short in simulating real-life graphs with high quality, as they tend to be tailored for specific graph families and lack the ability to learn directly from real-world graphs. With the development of deep learning, recent researches tend to simulate real-life graphs through deep learning techniques such as recurrent neural network (RNN) [14], [17] and generative adversarial network (GAN)(GAN) [15], [18]. These sophisticated models have notably enhanced the quality of graph simulations. However, they do come with their own set of limitations.

(1) Community-preserving. Because of the complexity of graph similarity computation problem, it is challenging to assess the quality of graph simulation directly through similarity score between two graph distributions. [1] Therefore, multiple metrics are employed to quantitatively measure similarity of graph distributions from different perspectives, such as clustering coefficient distribution and degree distribution. However, community structure, a key and distinctive characteristic of graphs, is often overlooked by many graph generators. Although generators like Stochastic Block Models (SBM) [21] (as well as its variants DCSBM [22] MMSB [11] and SBMGNN [16]), BTER [23] and Chung-Lu model [24] take community structure into consideration, limited parameters restrict them to accurately capture community structure observed in real-life graphs. The significance of community structure lies in its ability to preserve the graph's high-order structural properties, which are crucial for downstream data analysis tasks like node classification and link prediction. For example, communities in actual networks might indicate social

Sheng Xiang and Ying Zhang are with the Australia Artificial Intelligence Institute, the University of Technology Sydney, Sydney, Australia.
E-mail: {sheng.xiang, ying.zhang}@uts.edu.au
Chenhao Xu is with the School of Computer Science, Peking University, Beijing, China.
E-mail: xuchenhao@stu.pku.edu.cn
Dawei Cheng is with the Department of Computer Science and Technology, Tongji University, Shanghai, China.
E-mail: dcheng@tongji.edu.cn
(Corresponding author: Dawei Cheng.)
Manuscript received 21 Apr. 2024, revised 22 Nov. 2024

[1]Note that although measures like graph edit distance [19] and maximum common subgraph [20] can be used to assess similarity between two graphs, they lack the ability to establish a bijective relationship between the metric and the graph distribution. As a result, it is impossible to determine whether two graphs originate from the same distribution with existing measures.

groups [25] while in guarantee-loan networks [26], [27], they might indicate financial institutions groups and dense lending relationships (cf., Figure 1 for an example). Community structure helps us understand and leverage networks [28]. Therefore, apart from existing metrics for evaluating graph simulation quality, the ability to preserve community structure of real-life graphs should be taken into consideration.

(2) Efficiency. Deep learning approaches to general graph generative models, unlike traditional methods, often require significant computational resources (such as GPU memory usage and time consumption) and are predominantly suited for small to medium-sized graphs due to their inherent complexity. Notably, models such as GraphRNN [14], GRAN [29] and BiGG [30] employ RNNs to construct complete adjacency matrices, while NetGAN leverages random walks to assemble graphs. These methods generally exhibit a computational complexity during training and inference of $O(b \times n^2)$, with n representing the number of nodes in the graph and b the number of training epochs. Considering the trade-off between efficiency and the quality of generated graphs, there is a pressing need for developing a new deep-learning framework that effectively balances scalability with the fidelity of graph simulation.

(3) Scalability. Although learning-based graph generation methods generate relatively high-quality graphs, they tend to be less scalable due to the high complexity of the learning-based models. When generating graphs with millions of nodes, most of them encounter bottlenecks in inference time, training time, and GPU memory consumption. For example, through experiments, as the scale of the original graph grows we found that most deep learning-based methods (e.g., MMSB [11], VGAE [31], NetGAN [15]) cannot generate graphs containing more than 100,000 nodes within a GPU memory limit of 24GB. Even if few methods (e.g., CondGEN-R [18]) limit the model size to 24GB, their training process consumes extremely high time cost, and the methods are thus not applicable. However, real-world application scenarios abound with large-scale graphs. There is an urgent need to design a deep learning method with good scalability while guaranteeing the quality of generated graphs.

Contribution. Modifying deep learning-based graph generators to maintain community structure of observed graphs is not straightforward because it's hard to efficiently integrate community-preserving feature without compromising simulation quality. Therefore, this paper focuses on developing a graph generative model that not only better preserves community structures but also performs well on efficiency.

Graph simulation tasks have achieved great success due to recent progress in generative adversarial networks (GANs) [15], [18], [32] and variational auto-encoders (VAEs) [16], [33], [34]. We persist in our investigation of this task and introduce the Community-Preserving Generative Adversarial Neural Network (CPGAN). This model is distinguished by its ability to maintain community structures and provides superior efficiency and scalability relative to conventional models. The architecture of CPGAN features a ladder network equipped with graph convolution and pooling layers, serving as a permutation-invariant encoder for graphs. For graph generation, CPGAN employs variational inference techniques on latent distributions. Additionally, its decoder (generator) uses a fully-connected network combined with a dot product method for predicting links. The encoder (discriminator) uses graph convolution and pooling layers to evaluate how effectively the community structure has been captured from the observed graphs. The main contributions of this paper are summarized as follows:

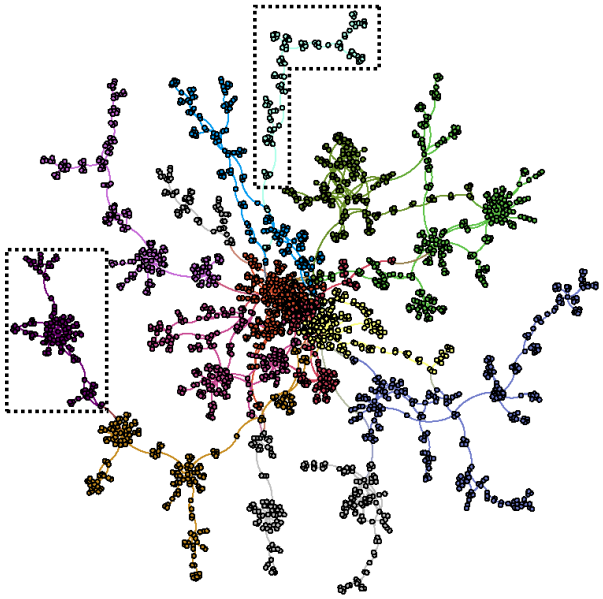

Fig. 1. An illustration of the communities (highlighted with dashed boxes) of a real-life network. Nodes with the same color belong to the same community

- We propose a Community-Preserving Generative Adversarial Network (CPGAN), a novel graph generative model that not only maintains the community structure and other key features of real-life graphs but also reach a trade-off between graph simulation quality and scalability. Furthermore, we introduce Scalable Community-Preserving Graph Attention Networks (SCPGAN), a short-pipeline and scalable version of the CPGAN, with similar generative performance and faster training speed.
- We have developed both generator and discriminator within a GAN framework. The generator, part of a hierarchical graph variational autoencoder, learns permutation-invariant representations of input graphs and then generates novel graphs from embeddings. Meanwhile, the discriminator evaluates if these embeddings originate from actual or simulated graphs.
- Comprehensive experiments on synthetic and real-world graphs reveal that our model effectively balances graph simulation quality and efficiency (scalability) in comparison to baseline methods. By experimenting with the community-preserving graph generation task, it is shown that SCPGAN performs better in generative quality than all models except CPGAN. And SCPGAN performs better than all other learning-based graph generators (including CPGAN) in scalability. Specifically, SCPGAN reduces the training time of the original CPGAN model by 4 times and the memory footprint by 4 times.

Roadmap. In Section II, we overview the graph generative models related to this paper. In Section III, We define the community-preserving problem formally and summarize the main challenges for community-preserving problem. Subsequently, we detail our proposed graph generative methods and outline their optimization objectives in Sections IV and V. Comprehensive experimental results for graph generative models and community-preserving experiments are presented in Section VI. Section VII make a conclusion of the paper.

II. Related Work

This section provides an overview of two primary areas: traditional graph generation methods (Section II-A) and deep graph generative models (Section II-B).

A. Traditional Graph Generation Methods

Graph Generation is one of important task of graph analytics [35], [36]. Graph generative models have a rich history of development, with seminal works including those by Erdős and Rényi [8], Albert and Barabási [9], and Leskovec et al. [13]. Classical models such as the Barabási-Albert (B-A) model [9], Chung-Lu model [24], Kronecker graphs [13], BTER [23], exponential random graphs [37], and stochastic block models [11] have been meticulously crafted to replicate specific types of graph structures. For example, the ERGM [37] is a probabilistic framework that determines the likelihood of edges based on node attributes. Nevertheless, it primarily captures a limited array of graph statistics. Kronecker graphs [13] utilize matrix multiplication to generate large-scale adjacency matrices, though they face challenges in representing a variety of graph structures. The BTER model [23] adjusts for average clustering coefficients and degree distributions through a dual-layer edge sampling method, also integrating community structures using an enhanced E-R graph approach. Similarly, SBM [21] and its derivatives, DCSBM [22] and MMSB [11], account for community structures; however, their effectiveness is often constrained by their simplified stochastic approaches. These models typically use a single parameter to define intra-community connections and another for inter-community connectivity probabilities.

B. Deep Graph Generative Methods

The landscape of graph generation has been reshaped in recent years by advancements in deep learning. Notable contributions include VGAE [31], DeepGMG [38], GraphRNN [14], Graphite [34], GRAN [29], and CondGen [18], which have all surpassed traditional approaches in terms of performance. Both Graphite [34] and VGAE [31] utilize variational autoencoders (VAE) with graph neural networks for the encoding and decoding processes, yet they struggle with handling multiple graphs as they assume a fixed vertex set. NetGAN [15] addresses efficiency by adopting graph random walk methodologies, although it is constrained to producing graphs of constant size. DeepGMG [38] applies graph neural networks to model the probabilistic interdependencies between nodes and edges effectively, capable of learning from any graph structure. However, generating a graph with m edges and n vertices alongside a graph diameter $Diam(\mathcal{G})$ results in a computational complexity of $O(mn^2Diam(\mathcal{G}))$. GraphRNN [14] introduces a sequential approach using recurrent neural networks for graph generation, but it lacks permutation invariance due to its dependency on specific node ordering. GRAN [29] enhances model scalability with an auto-regressive framework that iteratively generates nodes and edges, yet it fails to maintain permutation invariance. CondGen [18] manages to preserve permutation invariance by incorporating a Graph Convolutional Network (GCN) in its encoding process and handling generation tasks within an embedding space. Graph U-Net [39] employs node selection for graph representation through upscaling and downscaling techniques, but overlooks community structure during the learning phase. SBMGNN [16], a variation of the stochastic block model, utilizes graph neural networks to estimate parameters for overlapping stochastic blockmodels, although it doesn't directly solve issues related to community preservation, hence mirroring the community preservation performance seen in other deep learning-based graph generative models. Generative adversarial networks (GANs) [40] have been increasingly applied in graph-oriented tasks, including graph embedding [41], [42], semi-supervised learning [43], privacy preservation [44], and graph generation [14], [18]. It is essential in graph generation to utilize the structural insights from the dataset, especially for maintaining the community structures. While some models utilizing pooling strategies [39], [45] show potential in capturing community architectures, effectively reproducing these structures remains a significant challenge.

III. Background

This section outlines the fundamental aspects of graph generation, as described in Section III-A, and explores the primary challenges associated with generating graphs that preserve community structures, detailed in Section III-B.

A. Problem Definition

Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with V being the set of n vertices and $E \subseteq V \times V$ comprising m edges connecting pairs of vertices u and v, where each edge $e = (u, v) \in E$. The representation of graph G also includes an adjacency matrix $\mathbf{A} \in \{1, 0\}^{n \times n}$, reflecting its undirected nature through symmetry. Additionally, a node-feature matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ captures the features of the vertices, where d is the feature dimension. These symbols are summarized in Table I.

Problem Statement. The objective of graph generative models is to replicate the structural distribution of a given observed graph G, allowing the generation of new graphs $\{G'\}$ that mimic the structural properties of G.

While aiming for generated graphs to conform to the distribution of the observed graph, proving identical distributions is challenging due to the complex nature of graph structures [14]. Therefore, our evaluation employs specific metrics, as discussed in Section VI-A, to measure graph similarity, including aspects like degree distribution.

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, respectively
G, D	The generator and discriminator, respectively
$z^{(k)}$	The node features of the k-th level's community structure

Regardless of the specific model type, significant advancements have been made toward enhancing generation quality through community structure preservation. For instance, model-based approaches such as BTER [23] focus on capturing community structure explicitly to improve graph generation of E-R model. Similarly, learning-based methods like SBMGNN [16] leverage community-aware embeddings to enhance structural fidelity during generation. These approaches underline the importance of community structure preservation as a critical aspect of graph generation quality. Further insights into these evaluation metrics, particularly those pertaining to community structure preservation, are provided below.

Evaluation of Community Preservation. Our research prioritizes the preservation of community structures present in the training graphs, which we consider as the baseline truth. The aim is to ensure that generated graphs replicate these community structures accurately. The success of this effort is evaluated using two established metrics: Adjusted Rand Index (**ARI**) [46] and Normalized Mutual Information (**NMI**) [47]. An explanation of these metrics follows.

In the context of a graph with n nodes, initial community partitioning $Y_c = \{y_1, ..., y_c\}$ is set. Subsequently, a graph generated by the model presents a different community partition $X_r = \{x_1, ..., x_r\}$, challenging the model to preserve the original community structure.

Assuming there's a one-to-one node mapping between the two graphs, the similarity of their community partitions can be measured by *Rand Index* (RI), which is formulated as follows:

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

where TP (true positives) refers to the number of node pairs that are of the same cluster in both X_r and Y_c, while TN (true negatives) refers to the number of node pairs that are of different cluster in both X_r and Y_c. Similarly, FP (false positives) and FN (false negatives) refers to the number of incorrectly grouped node pairs.

However, the Rand Index has a limitation: it generally scores higher for graphs with fewer communities, as graph nodes have a greater chance of being randomly assigned to the same community. We can use the Adjusted Rand Index (ARI) to modify the likelihood of random community assignments. As illustrated in Figure 2, elements in the contingency table represent the number of common nodes n_{ij} between two community partitions x_i and y_j,$a_i = \sum_{j=1}^{c} n_{ij}$ and $b_j = \sum_{i=1}^{r} n_{ij}$. The ARI is calculated by the contingency table, which is formulated below:

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

The ARI allows us to quantitatively assess how closely the community structure of the generated graph matches that of the real graph. Another metric for community-preserving is Mutual Information (MI), whose formula is 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 represents the number of nodes. Practically, we use the Normalized Mutual Information (NMI) as our metrics to scale the results between 0 and 1.

B. Challenges

1) Community Structure Preserved Graph Generation: Inspired by hierarchical clustering and node representation, CPGAN and SCPGAN use a graph encoder to capture community structures at various levels as input to the discriminator. The output of encoder contains main information of the entire graph and serves as input to the discriminator. The decoder uses combined node representations from different community structure levels as the input, ensuring a more nuanced and effective graph generation.

2) Permutation-Invariance: A graph with n nodes can have $n!$ permutations. Without permutation-invariance, different permutations might lead to varied graph representations. To solve it, we design a permutation-invariant architecture, ensuring that every layer and objective function in our model is permutation-invariance. Specifically, the encoder and decoder must satisfy following conditions:

$$\begin{aligned} &\textbf{Encoder: } \mathcal{E}(PAP^T) = \mathcal{E}(A), \\ &\textbf{Decoder: } D(G(PZ)) = D(G(Z)), \end{aligned} \tag{4}$$

where $P \in \{0, 1\}^{n \times n}$ represents a permutation matrix, Z represents the random samples from prior distributions, $\mathcal{E}$ represents the encoder, G represents the generator and D represents the discriminator.

3) Scalability and Efficiency: The learning-based graph generative model excels in simulating high-quality graphs but struggles with efficiency and scalability, making it challenging to generate large real-world graphs. To address this, a new deep learning model that balances efficiency (scalability) and simulation quality is needed. Our approach samples nodes without replacement to create subgraphs during training, effectively achieving this balance between efficiency and quality.

IV. COMMUNITY-PRESERVING GRAPH GENERATION

In this section, we show our model's framework and implementation details, and introduce how to design the training process to reconstruct and generate new graphs.

	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. A contingency table of two community partitions $X_r = \{x_1, \ldots, x_r\}$ and $Y_c = \{y_1, \ldots, y_c\}$. n_{ij} denotes the number of nodes assigned to both community x_i and y_j. In the last columns and last row, a_i and b_j denote the number of nodes assigned to community x_i and y_j, respectively.

A. Community-Preserving Model Architecture

The CPGAN model is structured around three core components: the encoder ($\mathcal{E}$), the decoder/generator ($\mathcal{D}$ or G), and the discriminator (D). The discriminator's role is to determine whether a given graph is derived from actual datasets, while the generator is responsible for decoding community structures to either recreate existing graphs or create entirely new ones. During the generation process, samples are decoded from predefined distributions to produce graphs. For reconstruction, the encoder's parameters are shared with both the discriminator and generator. The objective is to reintroduce the newly generated graph into the system to deceive the discriminator.

In terms of graph structure, for a given graph $\mathcal{G}$ with an adjacency matrix A and a feature matrix X, its structural details are encoded using a ladder encoder. If the original graphs come with verified community labels Y_c, obtained from community detection algorithms (e.g., [48]), these are used to guide the node community assignments in the matrix X_r. The discriminator evaluates the authenticity of graphs by analyzing both the community data and the overall graph representation. Additionally, community structure information from the coarsened graphs is relayed back to the original nodes via a hierarchical message transmission mechanism, aiding in the reconstruction of graph structures. Node representations derived from prior distributions are also employed in the generation of new graphs.

B. Ladder Message Transmission Encoder

Our model features a ladder-shaped encoder which dynamically adjusts the pooling strategy and captures the community structure details effectively. The encoder inputs are $X \in \mathbb{R}^{n\times d}$, where each node has d features, and an adjacency matrix $A \in \{0,1\}^{n\times n}$. Node features X are obtained from the spectral embeddings of matrix A, denoted as $X = X(A)$, and initially, X is set to an identity matrix. The model employs a sequence of convolution and pooling layers to progressively coarsen the graph.

1) Graph Convolution and Pooling: We progressively coarsen graphs through a set of assignment matrices $S = \{S^{(l)} \in \mathbb{R}^{n_l \times n_{l+1}}, 1 \le l < k\}$, with n_l, n_{l+1}, and k representing the counts of input nodes, output nodes, and layers, respectively.

At each layer l, we compute the transmitted messages $Z^{(l)}$ and the corresponding assignment matrix $S^{(l)}$ as follows:

$$\begin{aligned} Z^{(l)} &= \sigma(GCN_{l,\text{embed}}(X^{(l)}, A^{(l)})), \\ S^{(l)} &= \text{softmax}(GCN_{l,\text{pool}}(Z^{(l)}, A^{(l)})), \end{aligned} \tag{5}$$

where σ is the ReLU activation function. $Z^{(l)} \in \mathbb{R}^{n_l \times d_l}$ represents the matrix of structural features, $X^{(l)} \in \mathbb{R}^{n_l \times d_{l-1}}$ is the feature matrix for cluster nodes at layer l, and $A^{(l)} \in \mathbb{R}^{n_l \times n_l}$ is the adjacency matrix for these nodes. Each GCN layer is tasked with extracting structural information and optimizing the pooling strategy. To prevent over-smoothing, PairNorm [49] is applied post each GCN operation.

Using $S^{(l)}$, we derive the coarsened adjacency matrix $A^{(l+1)}$ and the updated embeddings $X^{(l+1)}$ as follows:

$$\begin{aligned} A^{(l+1)} &= {S^{(l)}}^T A^{(l)} S^{(l)}, \\ X^{(l+1)} &= {S^{(l)}}^T Z^{(l)}, \end{aligned} \tag{6}$$

This approach enables us to obtain node representations across multiple hierarchical levels.

2) Graph Readout and Transposed Pooling: The process for reading out node features involves averaging the features of the i-th level coarsened graph, calculated as $s_i = \frac{1}{n_i}\sum_{j=1}^{n_i} x_{ij}$, where x_{ij} represents the feature of the j-th node at the i-th level. The cumulative graph representation s is formulated by concatenating these averaged features from all levels:

$$s = s_1 \oplus \ldots \oplus s_k, \tag{7}$$

where k denotes the total number of layers. This representation s serves as the input to the discriminator.

C. Graph Decoder

Following the extraction of node representations by the encoder, the decoder is utilized to calculate the likelihood of link formation between nodes. The link generation probability is defined by the following expressions:

$$\begin{aligned} p_\theta(A_{ij}|h_{k,i}, h_{k,j}) &= \sigma(g_\theta(h_{k,i})^T g_\theta(h_{k,j})), \\ p_\theta(A_{rec}|Z_{vae}) &= \prod_{i=1}^{n}\prod_{j=1}^{n} p_\theta(A_{ij}|h_{k,i}, h_{k,j}), \end{aligned} \tag{8}$$

where the function g_θ, a two-layer Multilayer Perceptron (MLP), processes community data extracted from the node features $h_{k,i}$, representing the i-th node's characteristics. The matrix $A_{rec} \in \mathbb{R}^{n\times n}$ indicates the predicted link probabilities. For practical training of the decoder on extensive graph datasets, a subset of nodes n_s (where $n_s \ll n$) is selected through degree-based sampling to create smaller subgraphs, simplifying the prediction matrix to $A_{rec} \in \mathbb{R}^{n_s \times n_s}$.

D. Opitimination

The target function of CPGAN is calculated as follows, which is a minimax game with value function $V(G, D)$:

$$\begin{aligned} \min_{\phi_G}\max_{\phi_D} V(D,G) =& \frac{1}{n}\sum_{i=1}^{n}\log(D(A_i)) \\ &+ \mathbb{E}_{p(Z_{vae})\sim q(\cdot|Z_{rec})}\log(1 - D(G(Z_{vae}))) \\ &+ \mathbb{E}_{p(Z_s)\sim\mathcal{N}(\cdot|\mathbf{0},\mathbf{I})}\log(1 - D(G(Z_s))), \end{aligned} \tag{9}$$

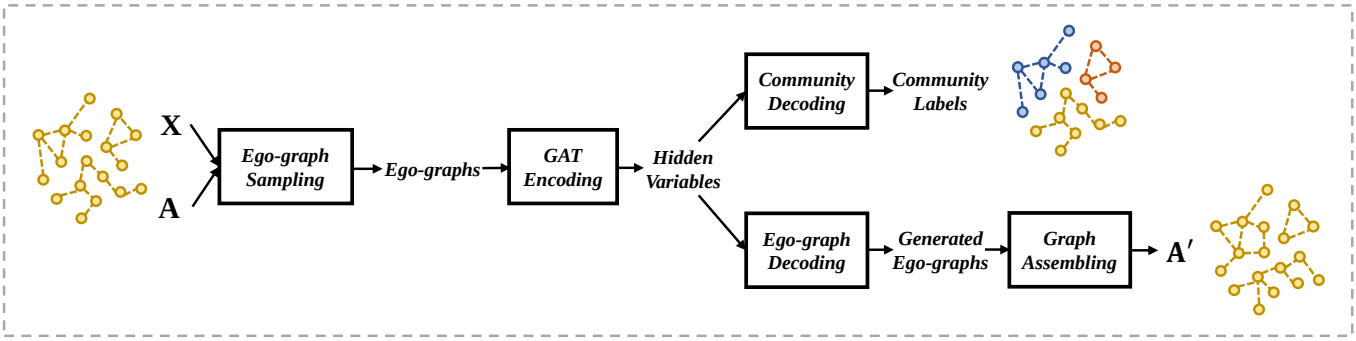

Fig. 3. The framework of SCPGAN.

where Z_{vae} is sampled from the approximate distributions, $D(A_i)$ is the MLP-based discriminator output, and Z_s is sampled from Gaussian prior distributions.

E. Training and Graph Generation

During training, the following steps are performed: (1) Subgraphs $A_{sub} \in \mathbb{R}^{n_s \times n_s}$ are sampled from the input graph, where n_s ($n_s \ll n$) represents the number of sampled nodes. These subgraphs are encoded into node embeddings using a graph encoder. (2) The generator takes the node embeddings as input and produces synthetic adjacency matrices A_{gen}. The generator is optimized to maximize the discriminator's error, ensuring that the generated graphs resemble real ones. (3) The discriminator is trained to accurately differentiate between real and synthetic subgraphs by minimizing a binary cross-entropy loss. This step ensures that the generator receives meaningful feedback for improving graph generation quality.

After training, we acquire $A_{sub} \in \mathbb{R}^{n_s \times n_s}$ by sampling n_s nodes and acquire the output matrix $A_{out} \in \mathbb{R}^{n \times n}$ which is acquired from the generator and is verified as a generated adjacency matrix by the discriminator. The matrix A_{out} is initially empty and is gradually filled with edges from each A_{sub} until the desired number of edges is reached. Determining each edge with a threshold and sampling through Bernoulli distributions parameterized by A_{out} result in potential exclusion of low-degree nodes and high-variance outputs, respectively. To mitigate them, the following approaches are adopted: (1) For each node i, an edge is generated by sampling from a categorical distribution parameterized by the i-th row of A_{out}. (2) The top-k entries of A_{out} are selected repeatedly until the predefined number of edges is met.

V. Scalable Community-Preserving Graph Generation

A. Prelimiaries

Although our proposed approach, CPGAN, is more efficient and scalable than other learning-based methods, it cannot generate very large graphs, such as those with 10M nodes. This limitation stems from the reliance of deep learning methods on GPU memory, which is smaller than CPU memory capacities. In contrast, large graphs can be more readily produced using traditional methods such as BTER [23] and ER [8]. For example, CPGAN requires that the entire graph fit within the GPU's memory for simulation, constraining its scalability. Nevertheless, CPGAN manages to process medium-sized graphs, up to 1M nodes, by decoding sub-graphs incrementally during each training iteration. Yet, this approach becomes impractical for graph sizes approaching 10M nodes due to the significant increase in memory demand.

B. Short-Pipeline Autoencoder Architecture.

To reduce parameters and computation costs, we drop the hierarchical ladder encoder and discriminator. Then, we keep the core encoder-decoder part: which composes the autoencoder-based framework of SCPGAN. In Figure 3, SCPGAN consists of three parts: graph attention encoder, community decoder, and graph decoder.

To avoid the whole graph computation in GCN [50], we leverage graph attention networks to measure the edge importance in the sampled local structure. Specifically, given input node features $\mathbf{H}_{in} \in \mathbb{R}^{n \times d_{in}}$ and ego graphs, we obtain the hidden variables $\mathbf{h}_u$ of the center node u of the associated ego-graph through leveraging the multi-head attention mechanism to aggregate messages from graph structures, where d_{enc} denotes the dimension of hidden variables after the encoding process. For each ego-graph, the message aggregating is formulated as follows:

$$\begin{aligned} \mathbf{h}_u =& \mathbf{GAT}_{enc}(\mathbf{X}|\tilde{V}_{ego}(u), \tilde{E}_{ego}(u)) \\ =& \text{Concat}(\text{Head}_1, ..., \text{Head}_{h_{attn}})\mathbf{W}_o \end{aligned} \tag{10}$$

where $\mathbf{h}_u \in \mathbb{R}^{1 \times d_{att}}$ denotes one row of hidden variables of the encoding layer, i.e., hidden variables on node u, and $\mathbf{W}_o \in \mathbb{R}^{h_{attn} d_{in} \times d_{att}}$ denotes the output projection matrix, h_{attn} denotes the number of attention heads, d_{att} is the dimension of the attention vector $\mathbf{a}$, and each head of graph attention layer $\text{Head}_i \in \mathbb{R}^{1 \times d_{att}}$ is formulated as follows:

$$\text{Head}_i = \sigma\Big(\sum_{v \in \mathbf{N}(u)} \alpha^i_{u,v} \mathbf{h}_u\Big) \tag{11}$$

where σ denotes the activation function, $\mathbf{N}(u)$ denotes the neighbor nodes of node u, and $\alpha^i_{u,v}$ denotes the importance of edge (u, v) in i-th head, which is formulated as follows:

$$\alpha^i_{u,v} = \frac{\exp(\text{LeakyReLU}(\mathbf{a}_i^T[\mathbf{h}_u || \mathbf{h}_v]))}{\sum_{k \in \mathbf{N}(v)} \exp(\text{LeakyReLU}(\mathbf{a}_i^T[\mathbf{h}_k || \mathbf{h}_v]))} \tag{12}$$

where $\mathbf{a}_i \in \mathbb{R}^{2d_{in}}$ denotes the attention vector of the i-th attention head, and LeakyReLU denotes the non-linear activation function with a negative input slope $\alpha = 0.2$.

Inspired by [51], we use the community labels to navigate the latent variables into *community-preserving latent space*.

Besides, to avoid $O(n^2)$ computation in the graph decoder, we drop the inner-product decoder. Then, we use a linear decoder to obtain the score of each edge. The two decoders are formulated as follows:

$$\begin{aligned}\bar{S} =&\text{softmax}(\mathbf{H}\times\mathbf{W}_{com}+\mathbf{b}_{com})\\ \bar{A} =&\text{softmax}(\mathbf{H}\times\mathbf{W}_{dec}+\mathbf{b}_{dec})\end{aligned} \tag{13}$$

where $\bar{S}$ denotes the predicted community assignment probability matrix, and $\bar{A}$ denotes the predicted edge probability matrix. In practice, the ground truth community labels of the nodes are calculated by the *louvain* [48] community detection algorithm. After generating the community labels of sampled nodes, we calculate the cross entropy loss from the ground truth community labels and update the community decoder and encoder parameters. Then, we generate the adjacency vectors for the sampled nodes and calculate the approximate loss from the real graph's adjacency matrix. The approximate loss function is formulated as follows:

$$\begin{aligned}\mathcal{L}_{SCPGAN} = -\frac{1}{n_s}(&\sum_{u\in\tilde{V}_s} A_u\log(\bar{A}_u)+\\ &\sum_{u\in\tilde{V}_s} Y_c^u\log(\bar{S}_u)\end{aligned} \tag{14}$$

where $\tilde{V}_s$ denotes the set of sampled center nodes, n_s denotes the size of $\tilde{V}_s$, $\bar{A}\in\mathbb{R}^{n_s\times n}$ denotes the score matrix from the graph decoder, Y_c denotes the ground truth community labels. By adjusting the value of n_s, we can achieve the trade-off between generating high-quality temporal graphs and fast model training. Assuming that we have sampled n_s nodes for model training (the details of sampling strategy is introduced in SecV-C), our proposed autoencoder architecture has a space complexity $O(n\times(n_s+d_{in}))$ and has $O(\frac{n}{n_s})$ time complexity for each training iteration.

C. Scalable Graph Sampling

To achieve a scalable graph generation and maintain the model's generative performance, we decompose the complete graph into numerous ego-graphs. And we achieve an approximation to the full-graph generation task by running our SCPGAN core architecture multiple times on the ego-graphs. Specifically, we first select representative nodes in a training epoch to model a complete graph structure. Nodes with higher degrees have a lower probability of being associated with outlier points. Therefore, to focus on representative nodes and edges and generate a high-quality graph, we use the probability distribution based on node degree as the initial node sampling strategy, which is formulated as follows:

$$P(u)=\frac{deg_u}{\sum_{v\in\tilde{V}} deg_v} \tag{15}$$

where deg_u denotes the degree of node u. Assuming that in each epoch, we sample n_s nodes as initial temporal nodes, we sample n_s ego-graphs as the input of our encoding process. The set of initial nodes is represented as $\tilde{V}_s$.

To reduce the time consumption of the encoding process, as illustrated in Figure 4 (a), we compute node representations on multiple ego-graphs for parallel node encoding to reduce computation steps from $O(n)$ to $O(\frac{n}{b})$, where b denotes the parallel number of temporal ego-graphs, i.e., batch size. For efficient training, we set the batch size as the size of initial sampled center node set with $b=|\tilde{V}_s|=n_s$. Therefore, the computation step is parallelized into $O(\frac{n}{n_s})$.

To further reduce the GPU memory space consumption, we use a truncation mechanism to control space usage and ignore repeated nodes when sampling nodes with replacement. As shown in Algorithm 31, we use th as the threshold to control the worst-case space requirement. Once a node's total number of neighbors exceeds th, the algorithm converts from all neighbor sampling strategy to th-neighbor sampling strategy.

To avoid repeated computation on high-degree nodes, after ego-graph sampling, as illustrated in Figure 4 (b), we merge all the ego-graphs into k-bipartite computation graphs. And we conduct message passing and edge importance computation on these computation graphs concurrently. As our sampled ego-graphs compose the representative structure of the observed overall graph, merging ego-graphs into bipartite computation graphs makes our graph attention networks able to efficiently

Algorithm 1: Sampling k-Radius Ego-Graph

```
Function NodeSampling(nodeset, threshold)
    Nodes← ∅; i, u ← 0;
    if length(nodeset)≤ threshold then
        return nodeset;
    else
        foreach i ∈ 1 : threshold do
            u ← random.choice(nodeset);
            Nodes.insert(u);
        return Nodes;
Function k-EgoGraph(G̃, v, k, th)
    ego, nodeset ← ∅;
    if k ≠ 1 then
        nodeset ← NodeSampling(N(v), th);
        foreach u^t ∈ nodeset do
            ego ← k-EgoGraph(G̃, u, k − 1);
            nodeset.insert(ego.nodes);
        ego ← G̃.subgraph(nodeset);
        return ego;
    else
        nodeset ← NodeSampling(N(v), th);
        ego ← G̃.subgraph(nodeset);
        return ego;
Function EgoGraphDataLoader(G̃, X, k)
    EgoGraphs, Nodefeatures ← ∅;
    ego, nodefeat ← ∅;
    foreach v ∈ 1 : n do
        ego ← k-EgoGraph(G̃, v^i, k);
        EgoGraphs.insert(ego);
        nodefeat ← X(ego.nodes, :);
        Nodefeatures.insert(nodefeat);
    return EgoGraphs, NodeFeatures
```

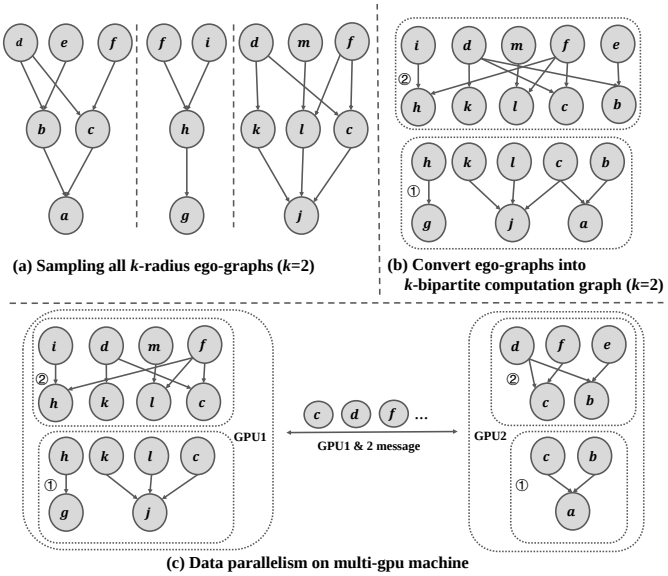

Fig. 4. Graph sampling result, composing computation graph, and data parallelism.

and effectively encode local structure representations.

D. Data Parallelism and Model Parallelism

We can easily run out of GPU memory when training on large graphs with more than one million nodes. Therefore, it is necessary to perform data-parallel training on multi-GPU machines. Suppose we have a dual-GPU machine. According to Figure 4 (c), we put the model parameters and the sampled ego-graphs into two GPUs, respectively. In the two GPUs, we assemble the respective ego-graphs into bipartite computation graphs. Then we calculate the representation of the center nodes, the predicted score matrices, and the gradient value of the parameter to be updated. Our ego-graph merging and encoder-decoder modules can be done completely in parallel. Only when there is a significant difference in the training time of the two GPUs will there be serial dependencies on updating embeddings of some nodes. For example, in Figure 4 (c), three node embeddings need to synchronize between two GPUs. According to our self-graph sampling strategy, we can limit the number of our embeddings below $O(t^k)$, where k is the sampled ego-graph depth and t is the size of the truncation value (i.e., threshold th in Algorithm 31), so the communication cost is controllable. With this data parallelism strategy, our SCPGAN has an upper bound of 2 billion nodes (i.e., $2^{31}-1$ nodes), which is far more than other learning-based methods.

We recommend using model parallelism if we need to generate huge graphs with more than 2^{31} nodes. Specifically, we divide the model into c copies and split the training data into multiple chunks. Each chunk contains less than 2^{31} nodes. Then we train the model on multiple clusters of machines. In this case, assuming that the depth of the self-graph we sample is 1, the time complexity of communication between machines is less than $O(t)$, where t is the truncation value in Algorithm 31, which has a manageable communication cost.

TABLE II
STATISTICS OF THE DATASETS USED IN THE EXPERIMENTS.

	#Nodes	#Edges	#Comm.	d_{mean}	CPL	GINI	PWE
Citeseer [52]	3,327	4,732	473	2.845	5.939	0.677	2.876
PubMed [52]	19,717	44,338	2,488	4.497	6.337	0.884	1.474
PPI [53]	2,361	6,646	371	5.820	4.376	0.743	1.903
3D Point Cloud [54]	5,037	10,886	1,577	4.322	32.40	0.828	1.928
Facebook [55]	50,515	819,090	8,010	32.43	14.41	0.716	1.503
Google [56]	875,713	4,322,051	9,863	9.871	6.378	0.673	1.825

E. Remarks and Discussions

As a solution, SCPGAN introduces a more scalable technique that extends beyond single-GPU limitations by utilizing multi-GPU, CPU memory, and even hard disk storage. Specifically, we (1) design an efficient graph autoencoder optimizing objectives for community-preserving graph generation; and (2) adopted an ego-graph sampling and bipartite computation graph assembling strategy, achieving a minibatch-based approach to train the graph generator; and (3) propose a data parallelism and a model parallelism architecture for scalable graph generative model training and inference.

VI. EXPERIMENTS

We conduct thorough experiments to confirm the effectiveness of our methods. Initially, we detail the experimental setup. Subsequently, we demonstrate the performance of our model in generating realistic graphs and preserving community structure in comparison with state-of-the-art baselines. Finally, we assess efficiency and memory usage of the model.

A. Experiment Settings

Datasets. We experiment with six representative datasets in the literature including two citation networks (Citeseer and PubMed) [52][2], PPI [53][3], 3D point cloud [54],[4] Facebook [55],[5] and Google web pages [56].[6] These datasets encompass various domains and exhibit diverse community structures. Table II displays the graph statistics of datasets, with "#Comm." indicating the number of communities. Detailed information of each dataset is shown as follows:

- **Citation Networks**. Citeseer and PubMed, two representative citation networks, comprises 3,327 and 19,717 publications, and 4,732 and 44,338 citations, respectively. Node refer to publication and edges refer to the citation relationships.
- **PPI**. The Protein-Protein Interaction (PPI) network is composed of 2,361 nodes and 6,646 edges. Each node refers to a yeast protein and each edge refers to interactions between two proteins.
- **3D Point Cloud**. The dataset has 5,037 nodes and 10,886 edges. Nodes refer to household objects and edges are generated for k-nearest neighbors, measured by the Euclidean distance of points in 3D space.

[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 IN COMMUNITY STRUCTURES BETWEEN THE GENERATED GRAPH AND THE REAL GRAPH, WHERE THE HIGHER IS BETTER.

Graph	Citeseer [52]		PubMed [52]		PPI [53]		3D Point Cloud [54]		Facebook [55]		Google [56]	
	NMI(%)	ARI(%)	NMI(%)	ARI(%)	NMI(%)	ARI(%)	NMI(%)	ARI(%)	NMI(%)	ARI(%)	NMI(%)	ARI(%)
SBM [21]	19.7±0.9	1.9±0.1	4.4±0.2	0.3±0.1	11.3±0.7	1.2±0.1	37.0±1.3	11.4±0.7	14.5±2.0	2.1±0.3	24.4±0.9	1.3±0.4
DCSBM [22]	27.1±0.8	1.7±0.1	18.9±0.2	0.3±0.1	18.6±0.8	1.8±0.3	37.3±1.4	11.5±0.8	17.5±1.5	1.9±0.3	29.4±0.6	5.7±0.5
BTER [23]	27.3±0.7	1.8±0.1	19.1±0.2	0.3±0.1	19.0±0.7	1.7±0.1	38.1±1.2	12.1±0.8	17.9±1.2	2.1±0.2	30.3±0.7	5.8±0.5
MMSB [11]	26.7±0.9	4.4±1.0	OOM	OOM	15.4±0.6	0.8±0.4	7.1±0.4	1.3±0.3	OOM	OOM	OOM	OOM
VGAE [31]	63.0±0.4	29.0±1.5	42.0±0.3	15.0±0.4	50.4±0.6	40.0±1.2	57.0±0.8	8.2±1.1	OOM	OOM	OOM	OOM
Graphite [34]	62.8±0.7	28.2±2.1	43.0±0.5	15.1±0.4	52.3±0.8	33.4±1.9	58.8±0.4	13.2±0.3	OOM	OOM	OOM	OOM
SBMGNN [16]	62.6±0.5	21.5±1.0	39.3±0.5	14.1±0.5	56.9±0.4	31.0±1.6	59.2±0.9	15.9±1.1	OOM	OOM	OOM	OOM
NetGAN [15]	57.9±0.5	20.1±0.3	OOM	OOM	55.2±0.5	30.2±0.3	67.4±0.9	37.8±2.6	OOM	OOM	OOM	OOM
CPGAN	**72.5±0.4**	**44.3±1.5**	**45.8±0.9**	**34.1±1.1**	57.0±0.7	**44.2±1.3**	**70.6±0.6**	**39.9±1.4**	**54.7±1.0**	**28.4±1.6**	**38.7±0.5**	**30.8±0.5**
SCPGAN	70.3±0.4	42.1±1.2	43.2±0.8	33.1±1.0	**58.0±0.8**	43.2±1.1	67.6±0.7	37.7±1.2	52.3±0.8	27.2±1.1	36.3±0.6	21.9±0.9

- **Facebook**. The dataset is a social network with 50,515 nodes and 819,090 edges. Nodes refer to users and edges refer to mutual likes among users.
- **Google**. The dataset is a web graph with 875,713 nodes and 4,322,051 edges. Node refer to web pages and edges refer to hyperlinks among them.

Compared Methods. We evaluate our method against traditional models and deep graph generative models. All baseline models learn features from a set of graphs and generate novel graphs. Traditional baselines encompass E-R [8], B-A [9], Chung-Lu [24], SBM [21], DCSBM [22], BTER [23], Kronecker [13], and MMSB [11]. Learning-based generative baselines encompass VGAE [31], Graphite [34], SBMGNN [16], GraphRNN-S [14], NetGAN [15], and CondGen-R [18]. It's worth noting that we opt for the scalable variants of GraphRNN and CondGen as baselines. We propose CPGAN and SCPGAN graph generative models in this paper.

Parameter Settings and Evaluation Metrics. In the experiments, algorithms and evaluating scripts are implemented with Python-3.6, PyTorch-1.8.1, CUDA-11.1, and GCC-4.8.5. The experiments are conducted on a machine equipped with Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz, 80 GB RAM and NVIDIA RTX 3090 with 24 GB memory and 10496 cuda cores. Each algorithm utilizes one CPU core and one GPU. Learning-based graph generation algorithms generally exhibit $O(n^2)$ complexity in both time and space While some methods reduce complexity using random walks, achieving high-quality generation often requires sampling a massive number of random walks, which can make the effective training complexity exceed $O(n^2)$. In contrast, SCPGAN significantly reduces the complexity through ego-graph sampling and mini-batch training. Specifically, the time complexity for SCPGAN during training and prediction is $O(n/b)$, where b is the batch size, and the space complexity is $O(n \cdot b)$. These complexities are substantially lower than traditional learning-based methods, making SCPGAN highly efficient and scalable. For practical implementation, we recommend setting the batch size to 32 for single-GPU systems to balance memory usage and training efficiency. For multi-GPU systems, the batch size can be increased proportionally to the available memory and computational power. During the experiment, the graph convolution hidden size of the ladder message transmission encoder is configured as 128. The learning rate and the graph pooling size is configured as 0.001 and 128 respectively. For SCPGAN, the batch size is configured as 32. For baseline models, we adhere to original hyperparameter settings. To measure the performance, we utilize various benchmark metrics as follows:

Deg.: Maximum Mean Discrepancy (MMD) of degree distribution, indicating the distinction of degree distributions between generated graphs and real graphs.

Clus.: MMD of clustering coefficient distribution, indicating the distinction of clustering coefficient distributions between generated graphs and real graphs.

CPL: The variance in characteristic path length between generated graphs and real graphs.

GINI: The distinction of GINI index, a common measure of inequality in a degree distribution between generated graphs and real graphs.

PWE: The distinction of power-law exponent between generated graphs and real graphs.

NMI and ARI: To assess the preservation of community structures, we compare the similarity of community structure in both observed and generated graphs. We utilize the *louvain* [48] community detection algorithm to make an evaluation. Assuming that graphs with identical community structures should yield similar detection outcomes, we employ Normalized Mutual Information (NMI) and Adjusted Rand Index (ARI), widely-recognized clustering metrics,[7] to measure community structures of generated graphs quantitatively.

B. Graph Generation

In our research, we experiment on various aspects including community structure preservation, generative distribution distance, and parameter sensitivity. We do representative experiments to demonstrate the effectiveness of our novel CPGAN and SCPGAN in graph generation. Due to constraints in page length, we omit certain experiments with analogous results.

Preserving Community Structure. In our experiment, we utilize the *louvain* [48] community detection algorithm to assess the preservation of community structure of the generated

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

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

Graph	Citeseer [52]					3D Point Cloud [54]					Google [56]				
	Deg.	Clus.	CPL	GINI	PWE	Deg.	Clus.	CPL	GINI	PWE	Deg.	Clus.	CPL	GINI	PWE
E-R [8]	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 [9]	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 [24]	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 [21]	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 [22]	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 [23]	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 [13]	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 [11]	2.98e-2	1.84e-2	17.9	0.173	0.186	0.339	2	25.9	0.234	13.7	OOM	OOM	OOM	OOM	OOM
VGAE [31]	0.123	3.78e-2	18.2	0.477	0.126	0.731	1.96	30	0.864	13.8	OOM	OOM	OOM	OOM	OOM
GraphRNN-S [14]	1.34e-3	**1.48e-3**	17.3	7.32e-2	0.176	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
CondGen-R [18]	8.42e-2	0.14	20.8	0.362	0.295	0.604	1.73	30.4	0.658	14.1	OOM	OOM	OOM	OOM	OOM
NetGAN [15]	**1.07e-3**	1.51e-3	16.5	0.136	0.154	0.415	1.72	26.3	0.542	14.6	OOM	OOM	OOM	OOM	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**
SCPGAN	2.37e-3	4.65e-3	15.8	0.102	0.129	0.601	1.83	24.3	0.593	13.6	0.321	0.882	15.9	0.101	1.03

TABLE V
PERFORMANCE COMPARISON FOR GRAPH RECONSTRUCTION TASKS IN THE PPI AND CITESEER DATASETS.

Graph	PPI [53]							Citeseer [52]						
	Deg.	Clus.	CPL	GINI	PWE	Train NLL	Test NLL	Deg.	Clus.	CPL	GINI	PWE	Train NLL	Test NLL
VGAE [31]	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 [34]	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 [16]	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 [18]	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**
SCPGAN	8.78e-2	0.647	11.56	**7.32e-2**	0.476	2.12	4.19	9.22e-2	0.511	**1.04**	2.29e-2	1.39	2.52	3.87

graphs. This involves measuring the similarity of the community detection outcomes between the original and generated graphs. Table III presents the effectiveness of various methods in maintaining community structure, a primary objective of this study. We excluded certain baseline methods as they lack node permutation invariance. Additionally, a few algorithms are unable to simulate certain graphs because of memory limitations, and these results are indicated as "OOM" (Out of Memory) in the table.

The first 8 rows represent outcomes of baseline methods. Notably, CPGAN performs best overall, especially in ARI evaluation. Although BTER performs best among traditional baselines, it is uncompetitive with learning-based models. As mentioned in Section II-B, SBMGNN doesn't use deep neural network to preserve community structure, thus uncompetitive with other learning-based models.

Generative Distribution Distance. Table IV displays the efficacy of different graph generation methods. Each metric used highlights the differences between the graphs generated by these methods and actual graphs. The initial 12 rows summarize the results from baseline methods. From the first six rows in the table, it is evident that BTER outperforms other traditional graph generators. Compared to its peers, CPGAN not only excels in maintaining community structures but also ranks highly in other quality metrics, particularly for larger graphs. For graphs of considerable size, such as those in the 3D Point Cloud and Google datasets, CPGAN shows substantial enhancements. Specifically, CPGAN achieves one-fifth of the top scores in Citeseer, three-fifths in 3D Point Cloud, and leads with the highest scores in the Google dataset. Additionally, CPGAN secures the top position in both the PubMed and Facebook datasets. It is notably superior in the 3D Point Cloud and PPI datasets when compared to competing models. It is important to mention that for the PubMed, Facebook, and Google datasets, the complex memory requirements of traditional learning-based models frequently lead to memory overflow problems.

TABLE VI
TIME CONSUMPTION (SECONDS) PER GRAPH GENERATION.

#Nodes	0.1k	1k	10k	100k
E-R [8]	$\mathbf{4.6}e^{-4}$	$9.0e^{-3}$	0.46	10.1
B-A [9]	$1.0e^{-3}$	$1.2e^{-2}$	**0.11**	1.17
Chung-Lu [24]	$7.2e^{-4}$	$2.5e^{-3}$	0.18	2.38
SBM [21]	$6.1e^{-3}$	0.09	2.58	37.1
DCSBM [22]	$6.2e^{-3}$	0.09	2.69	39.3
BTER [23]	$1.28e^{-3}$	$\mathbf{1.9}e^{-3}$	0.16	**0.25**
MMSB [11]	$6.1e^{-3}$	0.09	2.56	-
Kronecker [13]	$8.5e^{-3}$	0.08	1.00	9.69
GraphRNN-S [14]	0.27	4.74	63.6	-
VGAE [31]	$4.2e^{-3}$	0.04	0.38	-
Graphite [34]	$6.1e^{-3}$	0.06	0.64	-
SBMGNN [16]	0.01	0.11	1.18	-
NetGAN [15]	$8.7e^{-3}$	0.09	1.12	-
CondGEN-R [18]	$8.3e^{-3}$	0.15	-	-
CPGAN	$9.1e^{-3}$	0.08	0.95	86.1
SCPGAN	$9.4e^{-3}$	0.16	1.14	9.07

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

#Nodes	0.1k	1k	10k	100k
MMSB [11]	0.11	0.91	40.3	-
Kronecker [13]	1.39	1.55	3.25	**4.73**
GraphRNN-S [14]	1.63	15.4	161	-
VGAE [31]	0.06	0.42	9.75	-
Graphite [34]	0.07	0.47	10.6	-
SBMGNN [16]	0.08	0.63	12.4	-
NetGAN [15]	0.27	2.80	31.1	-
CondGEN-R [18]	0.18	25.3	-	-
CPGAN	0.35	0.70	6.39	32.9
SCPGAN	**0.01**	**0.09**	**0.89**	9.37

TABLE VIII
PEAK GPU MEMORY USAGE (MB) DURING TRAINING.

#Nodes	0.1k	1k	10k	100k
MMSB [11]	**1575**	**1709**	18529	OOM
GraphRNN-S [14]	1913	1959	5501	OOM
VGAE [31]	1719	1759	4799	OOM
Graphite [34]	1719	1761	4819	OOM
SBMGNN [16]	1719	1767	5243	OOM
NetGAN [15]	2237	2552	5008	OOM
CondGEN-R [18]	1722	1789	-	-
CPGAN	1728	1760	2467	7930
SCPGAN	1718	1720	**1752**	**1956**

C. Graph Reconstruction

In this research, we conduct experiments using the PPI and Citeseer datasets. For these tests, 80% of the edges are used for training while the remaining 20% serve for testing, enabling the reconstruction of the full graph using the models. We evaluate the performance of the models based on the negative log-likelihood (NLL) obtained from the discriminator's scores, averaging across both the training and testing datasets. The results are documented in Table V. The data shows that our model is highly competitive in the PPI dataset and secures the top performance in the Citeseer dataset, markedly surpassing other models such as VGAE, Graphite, SBMGNN, and CondGen. These findings align with those from our graph generation studies, underscoring the robustness of our approach in graph generation tasks. It is noteworthy that GAN-based models, including CPGAN, have weak performance in maintaining CPL on graphs with low CPL values. Specifically, in the PPI dataset, where the CPL is recorded as 4.38 in Table II, GAN-based models demonstrate a notable deficiency in CPL preservation. The reason is that graphs with low CPL values tend to have dense edges. Dense graphs complicate the discriminator, compelling the generator of GAN to make more complicated decisions. This complexity leads to instability of the the structure of generated dense graph during adversarial training. As CPL is a sensitive measure for assessing the structure of generated graphs, GAN-based models tend to underperform on graphs with lower CPL values.

D. Model Efficiency

This subsection focuses on assessing the efficiency and scalability of graph generators. Table VI presents the time comsumption of a novel graph generation, with node counts ranging from 100 to 100,000. Table VII presents the time consumption of the entire training process as well as table VIII presents the peak memory usage during training. The results show that traditional graph generators, including BTER, Chung-Lu, E-R, B-A, SBM, DCSBM, and Kronecker are more efficient and scalable than learning-based models, particularly with larger graphs. Among learning-based models, SCPGAN demonstrates superior efficiency and scalability as graph size increases. Specifically, SCPGAN and CPGAN are the only models capable of dealing with graphs with 100K nodes in Tables VI, VII and VIII. Moreover, SCPGAN is the only model capable of dealing with graphs with 10M nodes in Table IX.

E. Evaluation on Scalable Model

For all the experiments above, we supplemented the corresponding results of SCPGAN to evaluate its performance in all aspects further. The extra findings are as follows:

As shown in Table III, CPGAN achieves the best community-preserving performance among all graph generators evaluated. Meanwhile, SCPGAN achieves a second-best performance compared with CPGAN on preserving community structure tasks. Besides, SCPGAN still achieves a better community-preserving performance than other baselines except CPGAN. Table IV shows the performance of different methods for aspects other than community-preserving in graph generation. It is shown that SCPGAN achieves competitive results in five measurements of the generated graphs on three datasets. Table V reports the results of graph reconstruction experiments. We can also find that SCPGAN outperforms all baseline models in most metrics and even surpasses CPGAN in individual metrics, with only a minimal quality loss.Efficiency and scalability are the focus of SCPGAN's improvements. Thanks to its autoencoder architecture and parallel ego-graph training strategy, SCPGAN significantly reduces the training time and GPU memory usage of CPGAN while ensuring the quality of the generated graphs, as shown in Table VI, VII, and VIII.

To further explore the efficiency and scalability of SCPGAN, we scale the test data to ten million nodes and observe the performance of various learning-based graph generators in terms of training time and GPU memory consumption. Table IX report the efficiency results of a single machine with one GPU. It is shown that SCPGAN first scales up the generation of graphs to 10 million nodes level among the learning-based methods, according to Table IX.

We also plot the trends of inference time, training time, and GPU memory consumption with the graph scale for SCPGAN. As shown in Figure 5, SCPGAN has linear complexity in terms of time and space consumption as the number of nodes increases, which confirms its excellent scalability. Besides, according to the medium part of Figure 5, the training time complexity exceeds linear when the number of nodes exceeds 10^7. That is because our experiments are conducted on one GPU with 10496 cuda cores. According to our data parallelism strategy, this issue is trivial to address through deploy our model on multi-gpu machines.

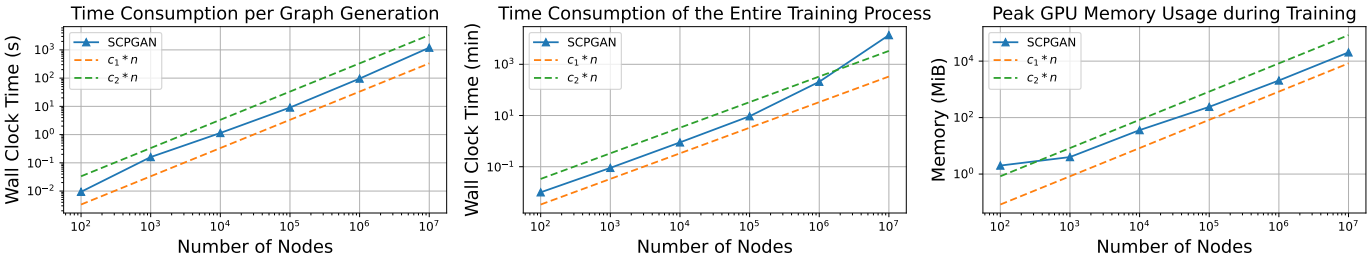

Fig. 5. Efficiency and scalability of SCPGAN.

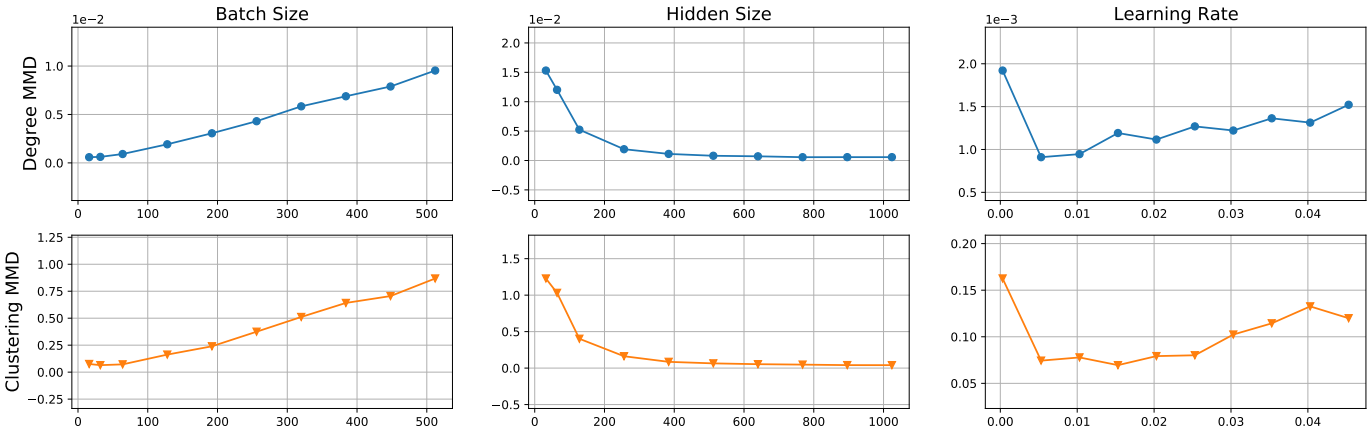

Fig. 6. Parameter sensitivity experiment results. Smaller points are better.

TABLE IX
CPU TIME CONSUMPTION (MINUTES) OF THE ENTIRE TRAINING PROCESS AND PEAK GPU MEMORY USAGE (MB) DURING TRAINING ON LARGE GRAPHS.

Nodes	1 Million		10 Million	
	Time	GPU	Time	GPU
MMSB [11]	-	OOM	-	OOM
GraphRNN-S [14]	-	OOM	-	OOM
VGAE [31]	-	OOM	-	OOM
Graphite [34]	-	OOM	-	OOM
SBMGNN [16]	-	OOM	-	OOM
NetGAN [15]	-	OOM	-	OOM
CondGEN-R [18]	-	OOM	-	OOM
CPGAN	1480	23879	-	OOM
SCPGAN	**208**	**3774**	**13802**	**22104**

F. Parameter Sensitivity

To further explore the properties of SCPGAN, we conducted parameter sensitivity experiments on the citeseer [52] dataset. According to the sensitivity experiment result, we selected the best hyper-parameters in our experiment. Figure 6 illustrates the maximum mean discrepancy (mmd) of node degree distribution and clustering coefficient between the generated and real graphs. SCPGAN performs best when the batch size is set as 32 because as batch size increases over 32, the quality of generated graphs decreases rapidly. The increase in hidden size makes SCPGAN more expressive and improves the performance of graph generation while increasing the GPU memory consumption. SCPGAN reaches the best performance when the hidden size increase to 400. Then the performance tends to remain as the hidden size increases over 400. Besides, the learning rate is preferably chosen to be between 0.005 and 0.01 because as the learning rate increases, the quality of generated graphs tends to increase before the learning rate of 0.005 and then decrease.

VII. CONCLUSION

In this paper, we introduced two deep generative models, CPGAN and SCPGAN, for simulating real-life graphs. CPGAN preserves community structure and other crucial graph properties. Although it shows superior performance in generating medium and large graphs compared to deep learning-based baselines, its efficiency and scalability for graphs over 10 million nodes remain limited. Consequently, we developed SCPGAN, a scalable variant of CPGAN, employing ego-graph sampling and data parallelism strategies. This approach enables training and inference on multiple GPUs and merges ego-graphs into bipartite computation graphs for efficient and effective local structure encoding. SCPGAN not only outperforms other learning-based methods in terms of efficiency and scalability as graph sizes increase but also maintains strong community preservation and competitive overall simulation quality. It presents the best scalability and a favorable balance between simulation quality and efficiency among learning-based graph generative methods.

ACKNOWLEDGMENTS

The work is supported by the National Key R&D Program of China (2022YFB4501704), the National Natural Science Foundation of China (62472317) and the Shanghai Science and Technology Innovation Action Plan Project (22YS1400600 and 22511100700) The authors want to thank Yang Duan 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] Y. Gao, X. Wang, X. He, H. Feng, and Y. Zhang, "Rumor detection with self-supervised learning on texts and social graph," *Frontiers of Computer Science*, vol. 17, no. 4, p. 174611, 2023. [Online]. Available: https://journal.hep.com.cn/fcs/EN/abstract/article_32720.shtml

[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 *dvances in Neural Information Processing Systems*. Curran Associates Inc., 2018, p. 6412–6422.

[5] A. Klipfel, Y. Frégier, A. Sayede, and Z. Bouraoui, "Optimized crystallographic graph generation for material science," in *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, E. Elkind, Ed. International Joint Conferences on Artificial Intelligence Organization, 8 2023, pp. 7145–7148, demo Track. [Online]. Available: https://doi.org/10.24963/ijcai.2023/836

[6] J. You, H. Wu, C. Barrett, R. Ramanujan, and J. Leskovec, "G2sat: Learning to generate sat formulas," *Advances in Neural Information Processing Systems*, 2019.

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

[8] P. Erdős, A. Rényi *et al.*, "On the evolution of random graphs," *Publ. Math. Inst. Hung. Acad. Sci*, vol. 5, no. 1, pp. 17–60, 1960.

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

[10] D. Watts and S. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, pp. 440–442, 1998.

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

[16] N. Mehta, L. Carin, and P. Rai, "Stochastic blockmodels meet graph neural networks," in *International Conference on Machine Learning*, 2019, pp. 4466–4474.

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

[18] 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.

[19] 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.

[20] V. Kann, "On the approximability of the maximum common subgraph problem," in *Annual Symposium on Theoretical Aspects of Computer Science*. Springer, 1992, pp. 375–388.

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

[22] B. Karrer and M. E. Newman, "Stochastic blockmodels and community structure in networks," *Physical review E*, vol. 83, no. 1, p. 016107, 2011.

[23] T. G. Kolda, A. Pinar, T. Plantenga, and C. Seshadhri, "A scalable generative graph model with community structure," *SIAM Journal on Scientific Computing*, vol. 36, no. 5, pp. C424–C452, 2014.

[24] 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.

[25] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99, no. 12, pp. 7821–7826, 2002.

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

[27] 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 *IEEE Conference on Visual Analytics Science and Technology*. IEEE, 2020, pp. 84–94.

[28] 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.

[29] 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.

[30] H. Dai, A. Nazi, Y. Li, B. Dai, and D. Schuurmans, "Scalable deep generative modeling for sparse graphs," in *ICML*, 2020, pp. 2302–2312.

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

[32] 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.

[33] M. Simonovsky and N. Komodakis, "Graphvae: Towards generation of small graphs using variational autoencoders," in *International Conference on Artificial Neural Networks*. Springer, 2018, pp. 412–422.

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

[35] M. Li, A. Micheli, Y. G. Wang, S. Pan, P. Lió, G. S. Gnecco, and M. Sanguineti, "Guest editorial: deep neural networks for graphs: theory, models, algorithms, and applications," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 4, pp. 4367–4372, 2024.

[36] K. Huang, Y. G. Wang, M. Li, and P. Lio, "How universal polynomial bases enhance spectral graph neural networks: Heterophily, over-smoothing, and over-squashing," in *Forty-first International Conference on Machine Learning*, 2024. [Online]. Available: https://openreview.net/forum?id=Z2LH6Va7L2

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

[38] 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.

[39] H. Gao and S. Ji, "Graph u-nets," in *international conference on machine learning*. PMLR, 2019, pp. 2083–2092.

[40] H. Wang, J. Wang, J. Wang, M. Zhao, W. Zhang, F. Zhang, X. Xie, and M. Guo, "Graphgan: Graph representation learning with generative adversarial nets," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Apr. 2018. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/11872

[41] 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.

[42] G. Y. Zihan ZHOU, Yu GU, "Adversarial network embedding using structural similarity," *Frontiers of Computer Science*, vol. 15, no. 1, p. 151603, 2021. [Online]. Available: https://journal.hep.com.cn/fcs/EN/abstract/article_26566.shtml

[43] 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.

[44] Y. SUN, G. LI, J. DU, B. NING, and H. CHEN, "A subgraph matching algorithm based on subgraph index for knowledge graph," *Frontiers of Computer Science*, vol. 16, no. 3, p. 163606, 2022. [Online]. Available: https://journal.hep.com.cn/fcs/EN/abstract/article_28363.shtml

[45] 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.

[46] D. L. Steinley, "Properties of the hubert-arabie adjusted rand index." *Psychological methods*, vol. 9 3, pp. 386–96, 2004.

[47] X. V. Nguyen, J. Epps, and J. Bailey, "Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance," *J. Mach. Learn. Res.*, vol. 11, pp. 2837–2854, 2010.

[48] 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, 2008.

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

[50] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=SJU4ayYgl

[51] P. Zhuang, O. O. Koyejo, and A. Schwing, "Enjoy your editing: Controllable {gan}s for image editing via latent space navigation," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=HOFxeCutxZR

[52] 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.

[53] 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.

[54] M. Neumann, P. Moreno, L. Antanas, R. Garnett, and K. Kersting, "Graph kernels for object category prediction in task-dependent robot grasping," in *Online proceedings of the eleventh workshop on mining and learning with graphs*, 2013, pp. 0–6.

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

[56] 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.

Sheng Xiang is a PhD candidate in the Center for Artificial Intelligence, major in Computer Science, University of Technology, Sydney (UTS). He received his BSc degree from Shanghai Jiao Tong University. His research interests include graph machine learning in finance, graph generative algorithms, bipartite graph processing, graph-based fraud detection, and dynamic graphs.

Chenhao Xu is currently a PhD student in the School of Computer Science, Peking University, Beijing, China. He received his BSc degree from Tongji University. His research interests include graph learning, big data, data mining and AI4DB.

Dawei Cheng is an associate professor with the department of computer science and technology, Tongji University, Shanghai, China. Before that, Dawei was a postdoctoral associate at MoE key lab of artificial intelligence, department of computer science, Shanghai Jiao Tong University. He received the Ph.D. Degree in computer science from Shanghai Jiao Tong University, Shanghai, China. His research fields include data mining, graph learning and big data in finance.

Ying Zhang is a Professor and ARC Future Fellow (2017- 2021) at Australia Artificial Intelligence Institute (AAII), the University of Technology, Sydney (UTS). He received his BSc and MSc degrees in Computer Science from Peking University, and PhD in Computer Science from the University of New South Wales. His research interests include query processing and analytics on large-scale data with focus on graphs and high dimensional data.