GraphGDP: Generative Diffusion Processes for Permutation Invariant Graph Generation

Han Huang[1], Leilei Sun[1]*, Bowen Du[1], Yanjie Fu[2], Weifeng Lv[1]
[1]*SKLSDE, Beihang University, Beijing, China*
[2]*University of Central Florida, Florida, USA*
Email: {h-huang, leileisun, dubowen, lwf}@buaa.edu.cn, yanjie.fu@ucf.edu

挑战

***Abstract*—Graph generative models have broad applications in biology, chemistry and social science. However, modelling and understanding the generative process of graphs is challenging due to the discrete and high-dimensional nature of graphs, as well as permutation invariance to node orderings in underlying graph distributions. Current leading autoregressive models fail to capture the permutation invariance nature of graphs for the reliance on generation ordering and have high time complexity. Here, we propose a continuous-time generative diffusion process for permutation invariant graph generation to mitigate these issues. Specifically, we first construct a forward diffusion process defined by a stochastic differential equation (SDE), which smoothly converts graphs within the complex distribution to random graphs that follow a known edge probability. Solving the corresponding reverse-time SDE, graphs can be generated from newly sampled random graphs. To facilitate the reverse-time SDE, we newly design a position-enhanced graph score network, capturing the evolving structure and position information from perturbed graphs for permutation equivariant score estimation. Under the evaluation of comprehensive metrics, our proposed generative diffusion process achieves competitive performance in graph distribution learning. Experimental results also show that GraphGDP can generate high-quality graphs in only 24 function evaluations, much faster than previous autoregressive models.**

***Index Terms*—Graph Generation, Generative Diffusion Process, Graph Neural Network**

I. Introduction

Graph data is a ubiquitous and highly applicable type of high-dimensional data in data mining. Modelling and understanding the generative process of graphs has been studied for a long time in network science [1] and continues as an active research topic incorporating deep learning [2]. Graph generative models aim to capture the underlying distributions over a particular family of graphs and generate diverse novel graphs with high fidelity, which serves as the foundation for wide applications, such as de novo drug discovery [3]–[6], computation graph creation for network architecture design [7], semantic parsing in natural language [8], and analysis in network science [9]–[11].

Learning the distribution of discrete and combinatorial graph structures is a challenging task, which is also a necessary and fundamental step for further jointly modelling attributes [4]–[6] and labels [12] in semantic abundant graphs. Traditional methods for graph generation date back to random graph models [9], [10], [13], which rely on hand-crafted stochastic generation processes and capture limited graph statistic properties. Recent deep graph generative models utilize the capacity of neural networks to learn graph structure distribution effectively. The prominent paradigms include variational autoencoder (VAE) based models [14]–[16], generative adversarial network (GAN) based models [17], [18], flow-based models [4]–[6], [19], and autoregressive models [3], [20]–[23]. Among them, autoregressive models achieve the most impressive generation quality on discrete graph structures. However, they rely on node generation orderings with high time complexity and fail to capture the important permutation invariant properties of graphs. The desired likelihood-based graph generative models should estimate invariant likelihood to all possible equivalent adjacency matrices of the same graph. To reach this goal, Niu *et al.* [24] creatively integrate score-based generative models [25] with graph neural networks to implicitly represent permutation-invariant distributions, but still suffer from generation quality and sampling speed.

In this paper, we propose continuous-time generative diffusion processes for permutation invariant graph generation (GraphGDP), which exhibit high graph generation quality and efficient sampling potential. Analogous to the diffusion processes in a non-equilibrium thermodynamic system in which particles move stochastically under the influence of a heat bath and spread out over the entire space in equilibrium [26]–[28], we perturb the edges in graphs with a sequence of noise (a.k.a, forward diffusion processes), and generate graphs by learning to reverse this process from noise to data (a.k.a, generative diffusion processes). It is non-trivial to adapt current methods to graph data effectively due to the permutation invariance constraint and the necessity to account both discrete local motifs and overall topological properties of graphs.

Inspired by the seminal work [29] with superior image generation quality, which connects diffusion-based models and score matching by defining the stochastic differential equation (SDE) describing continuous-time perturbing processes and the reverse-time SDE for generation, we define a forward diffusion process described by the specialized SDE with a closed-form expression over the real-valued adjacency matrices. We make an important observation that such a forward diffusion process implicitly defines a corresponding conversion process on the discrete distribution of graphs through a simple

* Corresponding author.
• Code is available at https://github.com/GRAPH-0/GraphGDP.

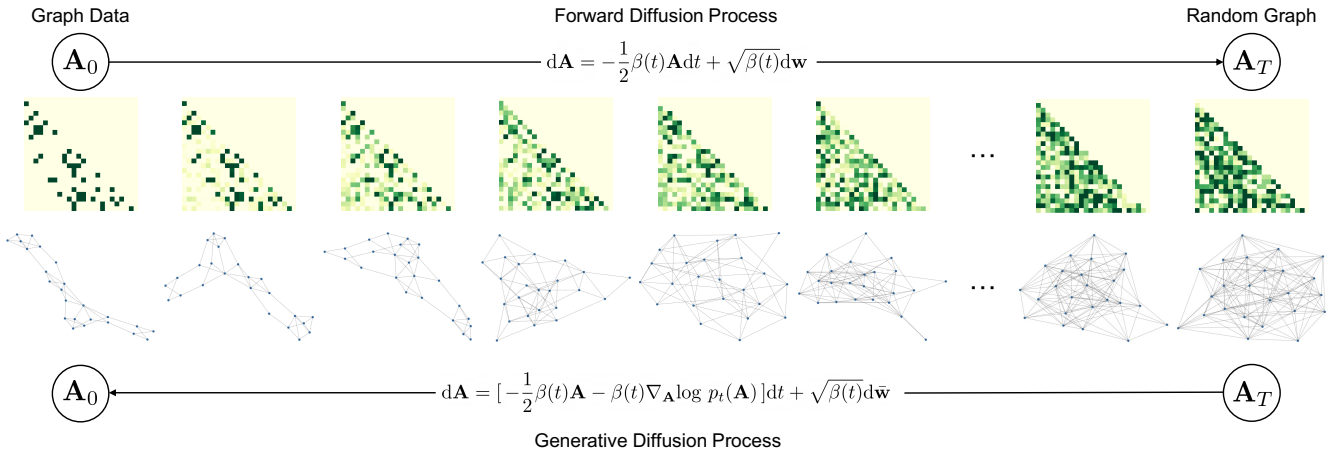


Fig. 1: Perturbing graph data to random graphs with an underlying simple edge distribution can be achieved by a continuous-time forward diffusion process described by an SDE. We can convert this SDE with the score of the data distribution $\nabla_{\mathbf{A}} \log\, p_t(\mathbf{A})$ at each time t . Thus, a graph sample can be transformed from a newly sampled random graph via the generative diffusion process. The lower triangle part of adjacency matrices perturbed from the same graph at different times are shown in the first row, and the corresponding discrete graphs are below them.

quantization on perturbed adjacency matrices. During the diffusion process, the edge existing probability in the Bernoulli distribution evolves stochastically over time, and the signal-to-noise ratio of the graph continues to decrease. In the final equilibrium state, the graph distributions are converted to Erdős-Rényi random graphs [13] with an edge probability of 0.5. The correlation between continuous-valued adjacency matrices and discrete graphs provides not only an intuitive understanding of the diffusion process on graphs, but also extra topology information for denoising in the reverse process.

Our ultimate goal is to recover the complex graph distribution from the prior noise distribution through generative diffusion processes. The success of reverse-time SDEs is heavily dependent on the neural networks (named score networks) to estimate scores (a.k.a., the gradient field of the perturbed data distribution). To capture the evolving graph topology, we emphasize the use of discrete graph states quantized from perturbed continuous-valued adjacency matrices. Intuitively, the evolving node or edge position information is critical in graph structure denoising. We propose an effective position-enhanced graph score network (PGSN) for permutation equivariant edge score estimation, which extracts structure and position features from quantized graphs and combines them with dense continuous adjacency matrices. With the optimized PGSN, we utilize numerical solvers for reverse-time SDEs and obtain final graph samples along approximate trajectories. For the graph generation quality evaluation, we apply metrics based on recent works [30], [31] to benchmark graph generative models on widely used datasets, and we find that our model achieves better or comparable performance to strong autoregressive models. Using the probability flow ordinary differential equation (ODE) sharing marginal probability densities with the SDE, we also show that our model can achieve efficient graph sampling with considerable quality in only 24 function evaluations using ODE solvers. The framework is summarized in Fig. 1.

贡献，创新点

Our main contributions are summarized as:

- We propose a novel continuous-time generative diffusion process for permutation invariant graph generation, which incorporates observations of continuous-valued adjacency matrices and corresponding discrete graphs for the inter-conversion between graph samples and random graphs.
- We design a position-enhanced graph score network that accurately estimates scores on perturbed graphs by leveraging the perturbed adjacency matrices as well as the structure and position information of discrete graphs.
- With the carefully designed evaluation setting, our proposed model achieves better or comparable sampling quality to autoregressive models in graph generation and displays strong potential for efficient sampling.

II. Preliminaries

A. Continuous-time Generative Diffusion Processes

For a datapoint $\mathbf{x} \in \mathbb{R}^d$, consider a forward diffusion process $\mathbf{x}_t$ defined by an Itô SDE:

$$\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\mathrm{d}t + \mathbf{G}(\mathbf{x}, t)\mathrm{d}\mathbf{w} \ , \tag{1}$$

with an indexed continuous time variable $t \in [0, T]$, a standard Wiener process $\mathbf{w}$ (a.k.a., Brownian motion) , a drift coefficient $\mathbf{f}(\cdot, t) : \mathbb{R}^d \to \mathbb{R}^d$ and a diffusion coefficient $\mathbf{G}(\cdot, t) : \mathbb{R}^d \to \mathbb{R}^{d\times d}$. According to [29], [32], [33], running backwards in time from T to 0 (*i.e.*, with negative dt), a corresponding reverse-time diffusion process inverting the above forward diffusion process can derived as:

$$\begin{aligned}\mathrm{d}\mathbf{x} = \{&\mathbf{f}(\mathbf{x}, t) - \nabla \cdot [\, \mathbf{G}(\mathbf{x}, t)\mathbf{G}(\mathbf{x}, t)^\top\,] \\ &- \mathbf{G}(\mathbf{x}, t)\mathbf{G}(\mathbf{x}, t)^\top \nabla_{\mathbf{x}} \log p_t(\mathbf{x})\}\mathrm{d}t + \mathbf{G}(\mathbf{x}, t)\mathrm{d}\bar{\mathbf{w}} \ ,\end{aligned} \tag{2}$$

where $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ is the score function of the marginal distribution over data $\mathbf{x}$ at time t and $\bar{\mathbf{w}}$ is a reverse-time standard Wiener process.

Song *et al*. [29] show that the reverse-time diffusion process can be converted into a generative model known as the continuous-time generative diffusion process. The SDEs commonly used for diffusion take the simple form for drift and diffusion coefficients with $\mathbf{f}(\mathbf{x}, t) = f(t)\mathbf{x}_t$ and $\mathbf{G}(\mathbf{x}, t) = g(t)\mathbf{I}$. With some specific designs for $f(t)$ and $g(t)$, the marginal and equilibrium density of the SDE approximates a Normal distribution at time T, *i.e*., $\mathbf{x}_T \sim \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$. [1] Initializing $\mathbf{x}_0$ with a sample from the complex data distribution, the state $\mathbf{x}_t$ gradually approaches equilibrium via Eq. (1). After training a time-dependent score network $\mathbf{s}_\theta(\mathbf{x}_t, t)$, *i.e*., the parametric score function, for estimating the score $\nabla_{\mathbf{x}_t} \log\ p_t(\mathbf{x}_t)$, we can synthesize data via the reverse-time SDE in Eq. (2). The denoising score matching [34] objective is modified for score estimation training as:

$$\min_\theta \mathbb{E}_t\{\lambda(t)\mathbb{E}_{\mathbf{x}_0}\mathbb{E}_{\mathbf{x}_t|\mathbf{x}_0}[\,||\mathbf{s}_\theta(\mathbf{x}_t, t) - \nabla_{\mathbf{x}} \log\ p_{0t}(\mathbf{x}_t|\mathbf{x}_0)||_2^2\,]\}\ , \tag{3}$$

where $\lambda(t)$ is a given positive weighting function. When $\mathbf{f}(\cdot, t)$ and $\mathbf{G}(\cdot, t)$ are affine, the transition kernel $p_{0t}(\mathbf{x}_t|\mathbf{x}_0)$ keeps a tractable Gaussian distribution [35], which helps compute the score target and perturbed data efficiently.

B. Permutation Invariant Generation

The goal of graph generative models is to capture the underlying distribution $p(G)$ over graph instances and then sample a new graph from the learned distribution. A graph G with n nodes is defined as $G = (V, E)$, where V, E correspond to the node set and the edge set. In this paper, we only consider the undirected graphs without self-loops and multi-edges. Given a node ordering $\pi = (\pi_1, ..., \pi_n)$, the graph G is determined by its adjacency matrix $\mathbf{A}^\pi \in \mathbb{R}^{n\times n}$. Therefore, the graph distribution can also be represented by marginalizing over all adjacency matrices. We omit the π in $\mathbf{A}^\pi$ if not emphasizing the node ordering. Since the graph distribution is inherently invariant to any node permutation, graph generative models are expected to estimate the same likelihood to equivalent adjacency matrices. Niu *et al*. [24] make a theoretical analysis that if the edge output of the score network is permutation equivariant, then the gradient of log-likelihood estimation $\mathbf{s}_\theta(\mathbf{A})$ is permutation equivariant and the implicitly defined log-likelihood function $\log\ p_\theta(\mathbf{A})$ is permutation invariant.

III. Model

Our approach aims to apply continuous-time generative diffusion processes to graphs, capturing the desirable permutation invariant graph distribution. Below, we first describe the forward diffusion process that perturbs graph instances towards random graphs in Section III-A. Then we illustrate our specialized position-enhanced graph score network for estimating graph scores (*i.e*., denoising perturbed graphs) in Section III-B. The detail of graph generative diffusion processes is provided in Section III-C.

[1] We only use the "variance-preserving" SDEs in this paper, while there are other SDEs with different Gaussian distributions in equilibrium.

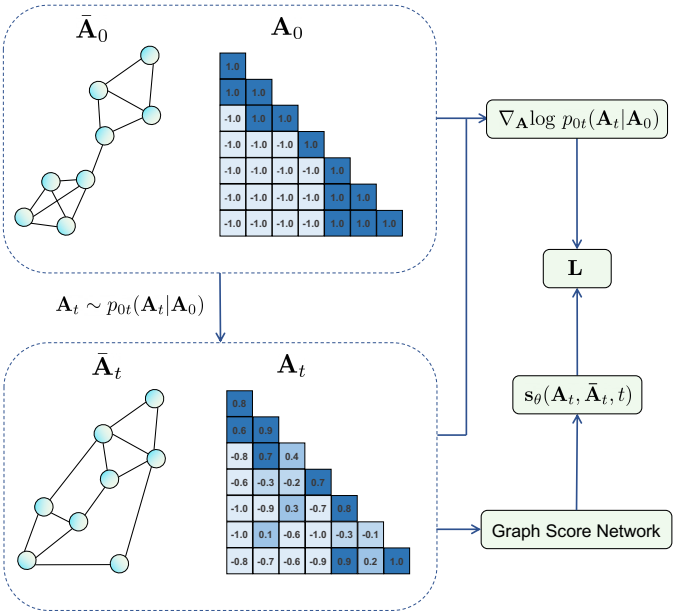


Fig. 2: The illustration of the objective computation for the proposed graph score network. Uniformly sample $t \in (0, 1]$, compute perturbed graphs and the gradient of log probability density at time t, and then train graph score networks to estimate scores.

A. Perturbing Graphs Towards Random Graphs

Our goal is to construct a diffusion process that perturbs graphs towards the equilibrium state like Eq. (1). Intuitively, a straightforward but effective method is to add noise into each element in adjacent matrices considering the discrete elements as continuous variables. For a graph diffusion process $\mathbf{A}_t$ indexed by a continuous time variable $t \in [0, T]$, we denote $\mathbf{A}_0 \sim p_0$ as an adjacency matrix drawn from the original data distribution p_0, and $\mathbf{A}_T \sim p_T$ as the final perturbed adjacency matrix drawn from the prior distribution p_T.

The choice of the drift coefficient function $f(t)$ and diffusion coefficient function $g(t)$ is indispensable for the success of diffusion process as shown in Section II-A. On the one hand, it should ensure that the corresponding SDE can diffuse the data from a complex high-dimensional distribution into a tractable prior distribution with a low signal-to-noise ratio that is convenient for sampling. On the other hand, it is assumed to be designed as affine functions that allow us to efficiently sample the state $\mathbf{A}_t$ at any time t in the forward diffusion process without model parameters. We apply the variance-preserving SDE introduced by [29] for graph data, whose discrete-time form is a denoising diffusion probabilistic model [28]. Considering the linear scaling function $\beta(t) = \bar{\beta}_{min} + t(\bar{\beta}_{max} - \bar{\beta}_{min})$ for $t \in [0, 1]$, we derive the SDE

from Eq. (1) as:

$$\mathrm{d}\mathbf{A} = -\frac{1}{2}\beta(t)\mathbf{A}\mathrm{d}t + \sqrt{\beta(t)}\mathrm{d}\mathbf{w} \ . \tag{4}$$

In detail, the values of $\mathbf{A}$ are linearly scaled to $[-1, 1]$. At time t, $\beta(t)$ actually control the scale of adding noise. $\beta(t)$ is shared for all elements in $\mathbf{A}$, while the noise term from $\mathbf{w}$ is independently sampled for each element. As graphs are undirected here, we manipulate the lower triangular part of adjacency matrices and make them symmetric afterwards. According to Eq. (4), with affine drift coefficients, the perturbation kernels $p_{0t}(\mathbf{A}_t|\mathbf{A}_0)$ are Gaussian distributions as [35]:

$$p_{0t}(\mathbf{A}_t|\mathbf{A}_0) = \mathcal{N}(\mathbf{A}_t; \mathbf{A}_0 e^{-\frac{1}{2}\int_0^t \beta(s)\mathrm{d}s}, \mathbf{I} - \mathbf{I}e^{-\int_0^t \beta(s)\mathrm{d}s}) \ . \tag{5}$$

With these perturbation kernels, we can perturb the graphs and compute the score target efficiently at any time without running the forward diffusion process step by step.

The forward diffusion process on graphs also implicitly defines a corresponding transformation on the discrete distribution, although it manipulates the continuous distribution. A simple quantization of the sampled $\mathbf{A}_t$ (*i.e.*, with the discretization threshold of 0.5 for the edge values) converts it to a graph denoted by $\bar{\mathbf{A}}_t$ that is sampled from the $\{0, 1\}^{n\times n}$ Bernoulli distribution with gradually erased original signals. The final equilibrium state $\bar{\mathbf{A}}_T$ represents the graph with adjacency matrix elements independently sampled from the Bernoulli distribution with an equal probability of having an edge or not, which is equivalent to sampling from Erdős-Rényi random graphs [13] with the edge probability of 0.5. From the perspective of $\bar{\mathbf{A}}_t$, it is actually a diffusion process on the discrete distribution. We emphasize the significance of leveraging the association with discrete graphs, which allows us to gain additional observations of intermediate graph states, such as changes in graph structure and position, while still benefiting from the convenient continuous SDE descriptions.

With the defined forward diffusion process including both continuous and discrete states, we build up the training procedure shown in Fig. 2 to supervise the learning of the graph score network. The training objective is defined as

$$\min_{\theta} \mathbb{E}_t\{\lambda(t)\mathbb{E}_{\mathbf{A}_0}\mathbb{E}_{\mathbf{A}_t|\mathbf{A}_0}[\,||\mathbf{s}_\theta(\mathbf{A}_t, \bar{\mathbf{A}}_t, t) - \nabla_{\mathbf{A}} \log\, p_{0t}(\mathbf{A}_t|\mathbf{A}_0)||_2^2\,]\}. \tag{6}$$

损失函数

We expect to utilize hybrid states for the graph score network to denoise the added Gaussian noise. At the beginning of diffusion processes, the node position, including the distance to other nodes, is quickly lost in real-valued adjacency matrices, since every element in the adjacency matrix is perturbed at the same time. Thus, if we estimate scores directly on dense adjacency matrices like EDP-GNN [24], such graph topology information will be absent, which is an important clue for graph structure representation learning. This motivates us to better design the graph score network.

B. Position-enhanced Graph Score Networks

A graph score network $\mathbf{s}_\theta(\mathbf{A}_t, \bar{\mathbf{A}}_t, t)$ learns to estimate the score of graphs with varying perturbation degrees, also known as a denoising model. Message passing based graph neural networks (GNNs) have become the de facto standard for graph structure representation learning, whose representation power is bounded by 1-WL test [36]–[38]. To enhance the representation power of GNNs, recent works [39]–[43] attempt to add the position encoding of nodes or edges to message passing architectures, including the position information from graph spectrum, random walks, and so on. Here, we propose a specialized **P**osition-enhanced **G**raph **S**core **N**etwork (PGSN) that utilizes graph structure and position features from intermediate discrete graphs, with real-valued adjacency matrices serving as initial edge features and contributing to score estimation through an edge feature updating mechanism.

We first extract appropriate node features and edge features from $\mathbf{A}_t$ and $\bar{\mathbf{A}}_t$. As the time information is added to all extracted features with the sinusoidal position embedding [44], we omit t in the description of PGSN. We take the degree onehot feature $\mathbf{h}^0$ as the initialization of node features, and obtain graph position information via the random walk $\mathrm{RW} = \bar{\mathbf{A}}\mathbf{D}^{-1}$ following [41], [43]. The node position feature consists of the landing probabilities of node i from the r-step random walks are defined as:

$$\mathbf{p}_i = [\,\mathrm{RW}_{ii}, \mathrm{RW}_{ii}^2, \cdots, \mathrm{RW}_{ii}^r\,] \ . \tag{7}$$

Using the same matrix of random walks, we also obtain the shortest-path-distance feature between node i and node j by

$$\mathbf{e}_{ij}^{spd} = \phi([\,\mathrm{RW}_{ij}, \mathrm{RW}_{ij}^2, \cdots, \mathrm{RW}_{ij}^r\,]) \ , \tag{8}$$

where $\phi(\cdot)$ takes the first non-zero position of the input vector and turns it to the onehot feature. Then the initial edge features are concatenated by $\mathbf{e}^0 = [\mathbf{A}W_0, \mathbf{e}^{spd}]$.

In order to capture the current graph structure thoroughly, we apply an L-layer message passing architecture incorporating node and edge features. The edge set is constructed by elements in $\mathbf{A}$ that are greater than the threshold γ (a hyperparameter controlling the computation burden and usually set to 0.2 in our experiments). We compute the two types of message for the node i at the l-th message passing layer as follows:

$$\alpha_{i,j}^{k,l} = \mathrm{softmax}\left(\frac{\mathbf{q}_i^{k,l}\ (\mathbf{k}_j^{k,l} \circ \mathbf{c}_{i,j}^{k,l})^\top}{\sqrt{d}}\right) \ , \tag{9}$$

$$\mathbf{m}_{i,j,(h)}^{k,l} = \alpha_{i,j}^{k,l}\ \mathbf{v}_j^{k,l} \circ \bar{\mathbf{c}}_{i,j}^{k,l} \ , \tag{10}$$

$$\mathbf{m}_{i,j,(p)}^{k,l} = \mathbf{m}_{i,j,(h)}^{k,l} \circ \mathbf{p}_j^l W_p^{k,l} \ , \tag{11}$$

where $\circ$ denotes element-wise multiplication. Here, $\mathbf{q}^l$, $\mathbf{k}^l$ and $\mathbf{v}^l$ are node features projected by different learnable matrices from the concatenation $[\mathbf{h}^l, \mathbf{p}^l]$, while $\mathbf{c}^l$ and $\bar{\mathbf{c}}^l$ are edge features projected from $\mathbf{e}^l$. The edge features not only bias the attention computation but also as a part of the aggregated

features. We aggregate and update the node embedding and the node position encoding by

$$\mathbf{M}^l_{i,(h)} = \Big\|_{k=1}^{H} \sum_{j\in N(i)} \mathbf{m}^{k,l}_{i,j,(h)} \ , \tag{12}$$

$$\hat{\mathbf{h}}^{l+1}_i = \mathrm{Norm}(\mathbf{M}^l_{i,(h)} + \mathbf{h}^l_i W_1) \ , \tag{13}$$

$$\mathbf{h}^{l+1}_i = \mathrm{Norm}\left(\hat{\mathbf{h}}^{l+1}_i + \mathrm{FFN}(\hat{\mathbf{h}}^{l+1}_i)\right) \ , \tag{14}$$

$$\mathbf{M}^l_{i,(p)} = \Big\|_{k=1}^{H} \sum_{j\in N(i)} \mathbf{m}^{k,l}_{i,j,(p)} \ , \tag{15}$$

$$\mathbf{p}^{l+1}_i = \mathbf{p}^l_i + \mathrm{act}(\mathbf{M}^l_{i,(p)} + \mathbf{p}^l_i) \ , \tag{16}$$

where $||$ is the concatenation of multi-head messages, Norm is a normalization layer, FFN is a two-layer feed forward network and act is an activation layer. Another important step is to update the edge feature after message passing as

$$\mathbf{e}^{l+1}_{i,j} = \mathbf{e}^l_{i,j} + \mathrm{act}\left((\mathbf{h}^{l+1}_i + \mathbf{h}^{l+1}_j)W_2\right). \tag{17}$$

Getting the final edge embedding $\mathbf{e}^L$, we concatenate it with the original $\mathbf{e}^0$ and adopt a multilayer perceptron (MLP) for the score estimation of each edge.

The standard message passing in graphs is theoretically guaranteed to be permutation equivariant [45]. Since the extracted node features and edge features are permutation equivariant, and since the operations in our PGSN consist purely of message passing and node-wise/edge-wise projections, the output edge scores are still permutation equivariant.

C. Graph Generation via Generative Diffusion Processes

After training with the time-dependent graph score network $\mathbf{s}_\theta$, we can construct the generative diffusion processes by the reverse-time SDE from Eq. (2) as

$$\mathrm{d}\mathbf{A} = [-\frac{1}{2}\beta(t)\mathbf{A} - \beta(t)\nabla_{\mathbf{A}}\log p_t(\mathbf{A})]\mathrm{d}t + \sqrt{\beta(t)}\mathrm{d}\bar{\mathbf{w}} \ , \tag{18}$$

where $\nabla_{\mathbf{A}}\log p_t(\mathbf{A})$ is the score function parametrized by $\mathbf{s}_\theta$. By first sampling from the prior Normal distribution (*i.e.*, sampling from random graphs), a new graph instance is generated by performing the generative diffusion process described by Eq. (18). Therefore, graph generation is transformed into a problem of numerically solving SDEs.

We utilize three numerical methods for solving the special reverse-time SDE, suitable for graph generation in different situations. First, many numerical solvers directly provide the approximate trajectory simulation of SDEs. For example, the Euler-Maruyama method is a simple discretization to the SDE, which is defined as

$$\mathbf{A}_{t-\Delta t} = \mathbf{A}_t + [\frac{1}{2}\beta(t)\mathbf{A}_t + \beta(t)\mathbf{s}_\theta(\mathbf{A}_t, \bar{\mathbf{A}}_t, t)]\Delta t + \sqrt{\beta(t)}\sqrt{\Delta t}\mathbf{z}_t, \tag{19}$$

where $\mathbf{z}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Given the number of discretization steps, we can determine Δt and generate graphs iteratively using the gradient information from the graph score network in Eq. (19). For simple and small graphs, such general SDE solvers provide adequate generation quality.

Second, for those graphs with complex structural characteristics (*e.g.* , large graph diameters), we can further employ Langevin MCMC [46] like score-based models [25], [29] to improve the sample quality at each discretization step. Specifically, after using Eq.19 to estimate the graph sample for the next step, we correct the estimated graph sample by

$$\mathbf{A}_t \leftarrow \mathbf{A}_t + \epsilon_t \mathbf{s}_\theta(\mathbf{A}_t, \bar{\mathbf{A}}_t, t) + \sqrt{2\epsilon_t}\mathbf{z} \ , \tag{20}$$

where the step size ϵ_t is determined by the norm of noise, the norm of scores and a hyperparameter r. The extra correction steps reduces the error from numerical solvers, obtaining more accurate margin distribution of graph samples.

Third, we can employ the corresponding ODE for efficient graph generation. [47] and [29] have explored the connection between generative diffusion processes and ODEs. Without the stochastic term in Eq. (18), we can derive the probability flow ODE [29], which corresponds to a deterministic process sharing the same marginal probability densities with the SDE. The probability flow ODE is defined as

$$\mathrm{d}\mathbf{A} = [-\frac{1}{2}\beta(t)\mathbf{A} - \frac{1}{2}\beta(t)\mathbf{s}_\theta(\mathbf{A}, \bar{\mathbf{A}}, t)]\mathrm{d}t \ . \tag{21}$$

It allows us to use current well-established ODE solvers to generate high-quality graphs in very few steps, which we describe in detail in Section IV-E. We quantize all the generated continuous adjacency matrices for final graphs.

IV. Experiments

In this section, we empirically demonstrate the power of the proposed GraphGDP in the task of graph generation.

A. Datasets 实验用到的数据集

We compare our graph generative model on four common graph datasets that vary in graph sizes and characteristics. (1) *Community-small*: 100 community graphs with $12 \le |V| \le 20$. The graphs are constructed by two communities with equal nodes, each of which is generated by the Erdős-Rényi model (E-R) [13] with $p = 0.7$. The inter-community edges are added with the uniform probability 0.05. (2) *Ego-small*: 200 one-hop ego graphs with $4 \le |V| \le 18$, extracted from Citeseer network [48]. The nodes represent documents and edges represent citation relationships. (3) *Ego*: 757 three-hop ego graphs with $50 \le |V| \le 399$, also extracted from Citeseer network [48]. (4) *Enzymes*: 563 protein graphs with $10 \le |V| \le 125$ selected from BRENDA database [49].

We further split the datasets into training and test sets with a ratio of $8:2$. The validation set comes from the first 20% of the training graphs. When evaluating the model performance on Community-small and Ego-small, we generate 1024 graph samples following [19], [24] to receive more stable evaluation results on small graphs. For Ego and Enzymes, we generate the same number of graphs as the test set.

B. Evaluation Metrics

Evaluating and comparing graph generative models is a challenging task, as it is difficult to obtain perceptual differences for graph visualization. We apply two type of metrics to comprehensively evaluate the quality of graph generation.

评估指标

1) Classical Structure Metrics: The widely-used evaluation metrics are based on Maximum Mean Discrepancy (MMD) measures to assess the distance between the distributions of the generated graph set $\mathbb{S}_g$ and the test set $\mathbb{S}_t$ [19]–[24]. Specifically, several graph property descriptor functions (*e.g.* degree distribution, clustering coefficient, 4-node orbit count histograms, and Laplacian spectrum) are applied to map each graph to high-dimensional representations. The estimate of MMD [50] on these representations can be derived as

$$\begin{aligned}\mathbf{MMD}(\mathbb{S}_g, \mathbb{S}_t) := \frac{1}{m^2}\sum_{i,j=1}^{m} k(\mathbf{x}_i^t, \mathbf{x}_j^t) + \frac{1}{n^2}\sum_{i,j=1}^{n} k(\mathbf{x}_i^g, \mathbf{x}_j^g) \\ - \frac{2}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m} k(\mathbf{x}_i^g, \mathbf{x}_j^t)\ ,\end{aligned} \tag{22}$$

where $k(\cdot,\cdot)$ is an optional kernel function, including a kernel using the first Wasserstein distance (EMD) or total variation distance (TV), and the radial basis function kernel (RBF).

Recently, O'Bray *et al.* [30] point out that the current practice of MMD metrics fail to faithfully reflect the distance of graph distributions. For example, MMD necessitates the use of positive definite kernel functions, and the previously-used hyperparameters fail to align with maximum discrimination in MMD. Following the suggestions from [30], we employ a more reasonable structure evaluation process to reflect the performance of graph generative models. **First**, we choose a valid and efficient kernel function, *i.e.*, an RBF kernel with a smoothing parameter $\sigma \in \mathbb{R}$ as $k(x_i, x_j) = exp(\frac{-\|x_i - x_j\|^2}{2\sigma^2})$. **Second**, we employ three graph-level structure descriptor functions, which are described in [20], [21], including (i) the degree distribution, (ii) the clustering coefficient distribution, and (iii) the Laplacian spectrum histograms. **Third**, we report the highest MMD values under a set of σ, expected to show the maximum distance between the two distributions. The 50 candidate values of $log\sigma$ are taken evenly at the interval in $[10^{-5}, 10^{5}]$. As for the number of bins used for the histogram conversion, we inherit the setting of [20], [21], which takes 100 bins for clustering coefficient and 200 bins for Laplacian spectrum. **Four**, MMD between the test and training graphs is included to provide a meaningful performance bound.

2) Neural-network-based Metrics: Thompson *et al.* [31] introduce several random GIN-based metrics for graph generative model evaluation, as the pre-existing structure metrics fail to capture the diversity of graph samples. The graph representations are extracted by random-initialized GIN [37], where MMD RBF (*i.e.*, MMD computed with the RBF kernel), F1 PR (*i.e.*, the harmonic mean of improved precision and recall) and F1 DC (*i.e.*, the harmonic mean of density and coverage) are built. MMD RBF is a more stable and comprehensive metric to measure the diversity and realism of generated graphs, while F1 PR and F1 DC are sensitive to detecting mode collapse and mode dropping. We follow the configuration of GIN in [31] and report the mean result of 10 random GINs.

C. Baselines

We compare the performance of our models against other graph generative models including VGAE [14], GraphRNN [20], GRAN [21], EDP-GNN [24] and BIGG [22]. For autoregressive graph generative models, we train models with the breadth-first-search (BFS) or depth-first-search (DFS) canonical node ordering schemes. In addition, we utilize uniformly distributed random node orderings to train several autoregressive models denoted by the extra $-\mathbf{U}$, which can be considered as an order-agnostic autoregressive model [51] that maximize the average likelihood over all node orderings of the graph [3], [23]. An Erdős-Rényi (**ER**) baseline [13] is also added, where the edge probability is estimated by the maximum likelihood over training graphs. The brief explanations and implementation details of deep graph generative models are listed as follows. **VGAE** [14] is a variational autoencoder that utilizes a graph convolution network encoder and a simple MLP decoder with inner product. **GraphRNN** [20] is an autoregressive model using a graph-level Recurrent Neural Network (RNN) to maintain graph states and another edge-level RNN to generate edges of the newly generated node. We re-train the officially implemented models with our dataset split using random BFS node orderings. **GRAN** [21] maintains the autoregressive process and utilizes graph neural networks with attention to model the graph generated context. GRAN generates a row of the adjacency matrix in a decision step to improve efficiency. The fixed DFS node orderings are utilized for Enzymes dataset, and the BFS node orderings for others. **BIGG** [22] is the state-of-the-art autoregressive tree-based model which utilizes the sparsity of realistic graphs. Over the sequence of nodes, it adopts a binary tree data structure to generate each edge and associates the set of edges with each node via a tree-structured autoregressive model. **EDP-GNN** [24] is a permutation invariance approach for graph generation via graph score matching and annealed Langevin dynamic sampling. We exploit its original model hyperparameters.

D. Graph Generation Quality

In this part, we benchmark the sample quality of our proposed graph generative model against other competitive models. For implementation details of our GraphGDP, the hidden dimensions are selected from 64, 128 and 256 for different datasets. We stack 4 message passing layers with 8 attention heads for graph score networks and adopt an MLP with 2 hidden layers for final score estimation. All our models are trained with Adam optimizer [52] and a constant learning rate $2e-5$. We also apply the exponential moving average (EMA) with the momentum 0.9999 for the parameter updating to improve stability as in [29]. Notably, the model parameters are shared across time, which is specified to the graph score network using the sinusoidal position embedding [44]. At the start of generation, we first sample the number of nodes based on the probability mass function of nodes in the training set. For datasets with high variance on graph sizes, we could also add the node number information to models as the condition like time information. We configure the variance preserve SDE

TABLE I: Comparison of the graph generation performance among graph generative models with classical structure metrics. The Train/Test shows the MMD results in a set of σ values between training and test graphs. For MMD metrics, the closer the value is to the Train/Test results, the better the performance. The top two cells in each column are coloured according to their rank. Deg.: degree distribution, Clus.: clustering coefficient distribution, Spec.: spectrum of graph Laplacian, Avg.: the average values of three MMD metrics.

	Community-small $\lvert V\rvert_{max}=20, \lvert E\rvert_{max}=62$ $\lvert V\rvert_{avg}\approx 15, \lvert E\rvert_{avg}\approx 36$				Ego-small $\lvert V\rvert_{max}=17, \lvert E\rvert_{max}=66$ $\lvert V\rvert_{avg}\approx 6, \lvert E\rvert_{avg}\approx 9$				Enzymes $\lvert V\rvert_{max}=125, \lvert E\rvert_{max}=149$ $\lvert V\rvert_{avg}\approx 33, \lvert E\rvert_{avg}\approx 63$				Ego $\lvert V\rvert_{max}=399, \lvert E\rvert_{max}=1071$ $\lvert V\rvert_{avg}\approx 145, \lvert E\rvert_{avg}\approx 335$			
	Deg.	Clus.	Spec.	Avg.	Deg.	Clus.	Spec.	Avg.	Deg.	Clus.	Spec.	Avg.	Deg.	Clus.	Spec.	Avg.
Train/Test	**0.035**	**0.067**	**0.045**	**0.049**	**0.025**	**0.029**	**0.027**	**0.027**	**0.011**	**0.011**	**0.011**	**0.011**	**0.009**	**0.009**	**0.009**	**0.009**
Order-dependent																
GraphRNN	0.106	0.115	0.091	0.104	0.155	0.229	0.167	0.184	0.397	0.302	0.260	0.320	0.140	0.755	0.316	0.404
GRAN	0.125	0.164	0.111	0.133	0.096	0.072	0.095	0.088	0.215	0.147	0.034	0.132	0.594	0.425	1.025	0.682
BIGG	0.041	0.073	0.050	0.055	0.024	0.029	0.028	0.027	0.020	0.019	0.019	0.019	0.034	0.108	0.077	0.073
Order-independent																
ER	0.300	0.239	0.100	0.213	0.200	0.094	0.361	0.218	0.844	0.381	0.104	0.443	0.738	0.397	0.868	0.668
VGAE	0.391	0.257	0.095	0.248	0.146	0.046	0.249	0.147	0.811	0.514	0.153	0.493	0.873	1.210	0.935	1.006
GraphRNN-U	0.410	0.297	0.103	0.270	0.471	0.416	0.398	0.429	0.932	1.000	0.367	0.766	1.413	1.097	1.110	1.207
GRAN-U	0.106	0.127	0.083	0.106	0.155	0.229	0.167	0.184	0.343	0.122	0.041	0.169	0.099	0.170	0.179	0.149
EDP-GNN	0.100	0.140	0.085	0.108	0.026	0.032	0.037	0.032	0.120	0.644	0.070	0.278	0.553	0.605	0.374	0.511
GraphGDP	0.039	0.074	0.052	0.055	0.023	0.029	0.030	0.027	0.023	0.025	0.019	0.022	0.037	0.099	0.021	0.052

TABLE II: Evaluation of different graph generative models using three neural-network-based metrics. The $50/50$ split represents the results computed with a random $50/50$ split of the dataset and shows the ideal scores for metrics. The top two cells in each column are coloured according to their rank.

	Community-small			Enzymes			Ego		
	MMD RBF (↓)	F1 PR (↑)	F1 DC (↑)	MMD RBF (↓)	F1 PR (↑)	F1 DC (↑)	MMD RBF (↓)	F1 PR (↑)	F1 DC (↑)
50/50 split	**0.037 ± 0.002**	**0.994 ± 0.012**	**1.065 ± 0.008**	**0.007 ± 0.000**	**0.988 ± 0.004**	**0.979 ± 0.006**	**0.005 ± 0.000**	**0.985 ± 0.004**	**1.025 ± 0.012**
Order-dependent									
GraphRNN	0.353 ± 0.088	0.252 ± 0.183	0.407 ± 0.171	1.495 ± 0.037	0.000 ± 0.000	0.000 ± 0.000	1.283 ± 0.053	0.019 ± 0.016	0.007 ± 0.007
GRAN	0.196 ± 0.014	0.824 ± 0.141	0.793 ± 0.099	0.069 ± 0.008	0.915 ± 0.035	0.738 ± 0.027	0.244 ± 0.064	0.238 ± 0.141	0.207 ± 0.088
BIGG	0.052 ± 0.003	0.135 ± 0.087	1.048 ± 0.035	0.019 ± 0.000	0.964 ± 0.008	0.966 ± 0.012	0.022 ± 0.002	0.956 ± 0.014	0.896 ± 0.026
Order-independent									
ER	0.278 ± 0.046	0.363 ± 0.201	0.335 ± 0.096	0.808 ± 0.065	0.046 ± 0.030	0.019 ± 0.005	0.118 ± 0.035	0.516 ± 0.116	0.377 ± 0.120
VGAE	0.360 ± 0.065	0.292 ± 0.165	0.292 ± 0.113	0.716 ± 0.033	0.012 ± 0.016	0.002 ± 0.003	0.520 ± 0.003	0.000 ± 0.000	0.000 ± 0.000
GraphRNN-U	0.970 ± 0.113	0.066 ± 0.043	0.079 ± 0.003	1.263 ± 0.177	0.000 ± 0.000	0.000 ± 0.000	1.317 ± 0.022	0.000 ± 0.000	0.000 ± 0.001
GRAN-U	0.164 ± 0.016	0.859 ± 0.082	0.888 ± 0.053	0.242 ± 0.033	0.671 ± 0.056	0.364 ± 0.024	0.128 ± 0.041	0.720 ± 0.041	0.564 ± 0.026
EDP-GNN	0.125 ± 0.004	0.913 ± 0.108	0.977 ± 0.044	0.119 ± 0.010	0.954 ± 0.012	0.846 ± 0.020	0.295 ± 0.061	0.395 ± 0.028	0.192 ± 0.036
GraphGDP	0.066 ± 0.012	0.656 ± 0.138	1.042 ± 0.014	0.026 ± 0.001	0.974 ± 0.005	0.932 ± 0.015	0.034 ± 0.004	0.877 ± 0.014	0.721 ± 0.023

with $\bar{\beta}_{min} = 0.1$ and pick $\bar{\beta}_{max}$ from $\{5, 10, 20\}$, ensuring that the signal-to-noise ratio at the last perturbed graphs is kept small. On Ego-small dataset, we apply Euler-Maruyama method with 1000 discretization steps for graph sampling, and we incorporate extra Langevin MCMC steps on other datasets.

The generation quality evaluation results with classical structure metrics are reported in Table I, where the smaller values of MMD metrics represent the smaller distance between the two distributions. Table II shows the performance using neural-network-based metrics. We colour the top two performance for each metric. Some visualizations of generated diffusion processes are shown in Figure 3.

We summarize the observations after analyzing the model performance for graph sample quality. (1) Among order-independent graph generative models, GraphGDP achieves remarkable performance improvement. (2) Under the same requirement to capture the permutation invariance property of graphs, the proposed GraphGDP outperforms the score-based EDP-GNN obviously, especially in larger graphs. (3) Compared to the dominant autoregressive generative models, GraphGDP surpasses the performance of competitive GraphRNN and GRAN for most metrics. Our method also shows better or comparable results to BIGG without using any predefined node orderings, except for slightly less diversity on the Ego dataset. In conclusion, GraphGDP demonstrates the high fidelity and diversity of generated graph samples on datasets with different characteristics.

E. Efficient Graph Generation

Sampling efficiency is a crucial property pursued by graph generative models [21], [22]. For generative diffusion processes, the connection between the reverse-time SDE and the deterministic probability flow ODE provides a way for fast sampling. We can generate graphs by solving the neural ODE described by Eq. 21. Through controlling the error tolerance of off-the-shelf adaptive-step solvers or the step size of fixed-step solvers [53], we generate high-quality graphs with many fewer steps (a.k.a., function evaluations). Compared to other domains, the elements in graph adjacency matrices are less

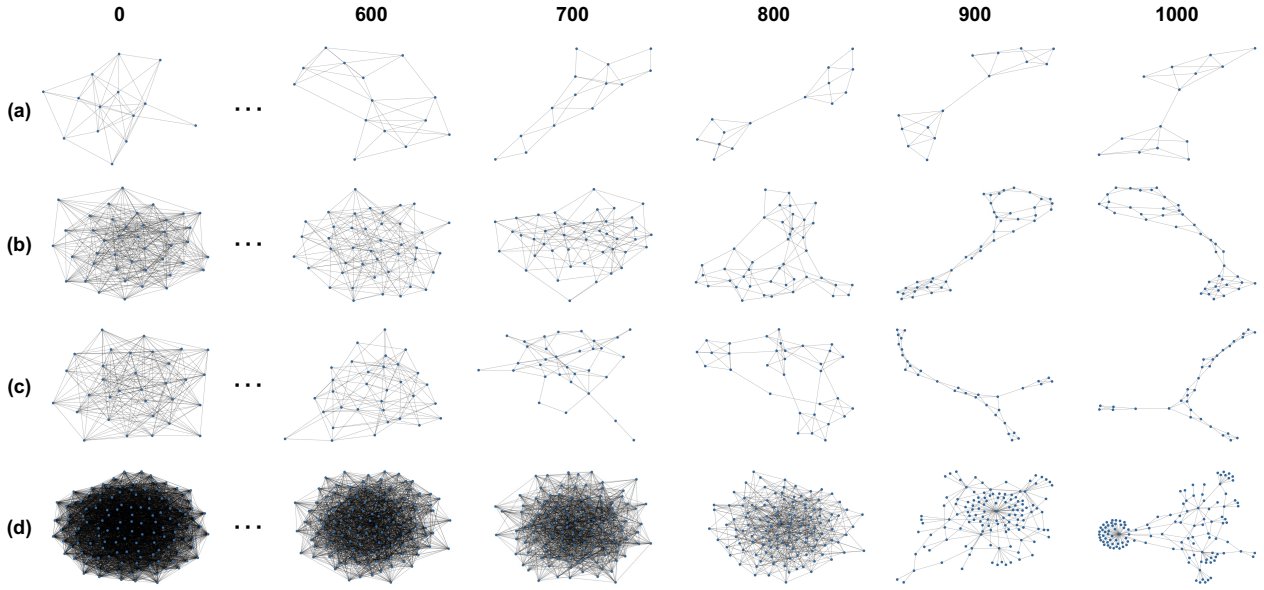
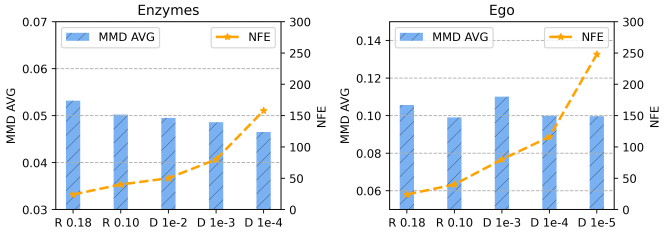
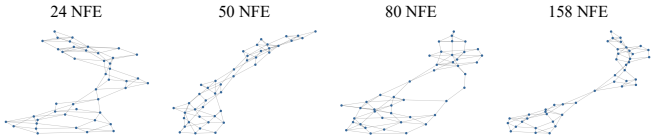


Fig. 3: Graph visualization of different steps in the generative diffusion processes on Community-small (a), Enzymes (b-c), Ego (d) datasets.



(a) Graph generation performance and NFE used by ODE solvers. MMD AVG: average values of three structure MMD metrics. R 0.18: "rk4" fixed-step solver with 0.18 step size. D 1e-2: "dopri5" adaptive-step solver with 1e-2 error tolerance.



(b) Graphs generated from the same latent code through ODE solvers with various NFE.

Fig. 4: Graph generation quality for the same model using different settings of the ODE solvers. The larger error tolerance or step size reduces the number of function evaluations (NFE), maintaining high generation quality.

informative with only 2 values, and may be more tolerant to errors, while the pixels in images have 256 values. As shown in Figure 4a, with different solver settings, our model keeps the structure fidelity with even **24** steps. The graph visualization in Figure 4b also shows that the overall structure patterns of the graph are maintained even if the graph topology changes slightly due to the numerical precision.

We compare the sampling quality and inference time of our efficient version with other models in Fig. 5. GraphGDP has clear performance and speed advantages compared to GRAN (designed for efficient sampling) and EDP-GNN. Another powerful model, BIGG, does not support graph generation in batch form, and is not put into Fig. 5 for comparison. On Ego dataset, our model takes on average 0.41s to generate a graph with one batch size, which is still faster than BIGG's 2.19s. In contrast to autoregressive models, graph generative diffusion processes have strong potential for efficient generation.

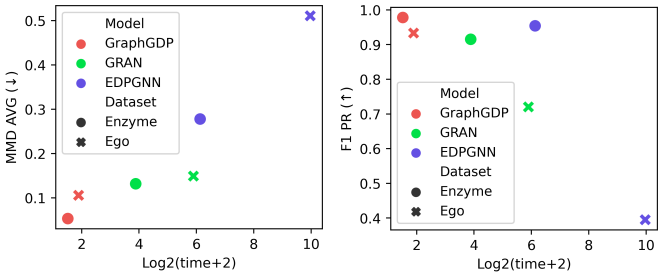


Fig. 5: Comparison of the log-scale running time for generating 16 graphs in a batch on Enzymes and Ego datasets.

F. Ablation Study

Utilizing generative diffusion processes, we compare PGSN with its variants and other graph score network parameterization methods on Enzymes dataset to validate the effectiveness of the design for graph score networks. All the models are trained with the unified 64 hidden dimensions and 1M training steps. The methods are denoted as follows:

- MLP: an MLP consisting of 3 hidden layers with perturbed graph adjacency matrices as input.

TABLE III: The results of various score function parameterization methods in generative diffusion processes on Enzymes dataset. RW: the steps of random walks in graphs.

Score Network	RW	Deg.	Clus.	Spec.	Avg.
MLP	-	0.703	1.044	0.213	0.654
GIN	-	0.331	0.468	0.071	0.290
GAT	-	0.365	0.479	0.071	0.305
GTN	-	0.158	0.457	0.074	0.230
PGSN w/o P	32	0.083	0.213	0.050	0.116
PGSN w/o U	32	0.107	0.212	0.048	0.122
PGSN	4	0.095	0.233	0.059	0.130
PGSN	8	0.097	0.222	0.054	0.124
PGSN	16	0.084	0.208	0.049	0.114
PGSN	32	**0.079**	**0.198**	**0.047**	**0.108**

- GIN: a 4-layer graph isomorphism network [37] that takes the degree onehot embedding as the initial node feature. The edge representations are obtained by adding the pair of node representations from the GIN. After concatenating the edge representations and original perturbed edge values, a 2-layer MLP outputs the estimated scores.
- GAT: a 4-layer graph attention network [54] with 8 attention heads which follows the same setup of GIN.
- GTN: a graph transformer network that modifies the graph attention networks with the dot-product attention and feed-forward network [40].
- PGSN w/o P: a PGSN variant without node position features consisting of landing probabilities of the node itself, but with the shortest-path-distance features.
- PGSN w/o U: a PGSN variant without updating edge features after message passing.

From the results in Table III, it can be observed that using node degree features and the shallow message passing architecture does help the generative models capture the overall degree distribution of the graphs, but fail to go further on the clustering coefficient metric which requires the more accurate graph local structure. Notably, introducing the shortest-path-distance features and learnable edge representations greatly improves the ability of the graph score network to denoise the perturbed graphs, while the node position feature from the landing probability of random walks also make contributions.

We also conduct parameter sensitivity analysis for the random walk steps, the results of which are included in Table III. Consistent with the intuition that more random walk steps yield more topology information of graphs, the model with more walks achieves better sample quality. Considering the computation cost of random walks, we recommend choosing an appropriate step number according to the characteristic of datasets to take into account both efficiency and effectiveness.

V. Related Work

In addition to the graph generation approaches mentioned before, we summarize the notable existing literature on the construction of our framework.

Generative Diffusion Processes. Diffusion probabilistic models [27] are inspired by non-equilibrium thermodynamics. By defining a Markov chain of diffusion steps to slowly add noise to data, they learn to reverse the inference path to generate data from the noise. Ho *et al*. [28] propose a simplified objective of diffusion models and connect it with noise-conditioned score networks [25] which use Gaussian noise to perturb data distribution over the full space. As it is relatively slow to generate a sample from the Markov chain of the generative diffusion process, a simple stride sampling schedule is proposed by [55]. Song *et al*. [47] define a deterministic generative process and generates high quality samples with a fewer number of steps. During the same period, Song *et al*. [29] propose a continuous-time generative diffusion process that takes advantage of the SDE and improves performance. The flexible model architecture and high sample quality on high-dimensional data attract us to adapt the continuous-time generative process for graph generation. Most recently, concurrent work [56] also studies diffusion models for graph generation but overlooks the discreteness of graphs.

Position-aware Graph Neural Networks. Position encoding plays a significant role in current neural networks like ConvNets and Transformers. For graph neural networks, position information of nodes or edges is also critical for graph structure representation learning [57], [58]. Adding unique or discriminating features to all nodes in graphs is one way to incorporate the node position [59]. But this type of position encoding lacks the generalization of unseen graphs. Laplacian positional encoding takes graph Laplacian eigenvectors that maintain global structure information as external node features [40], [42]. The existence of the sign ambiguity is the main weakness of Laplacian positional encoding. Random walk based position encoding [41], [43] reflects the graph topology by landing probabilities and path distances. Inspired by these position-aware GNNs, we realize that the position information in perturbed graphs can reflect the changes in graph structure, and design a position-enhanced graph score network.

VI. Conclusions

We propose a novel continuous-time generative diffusion process for permutation invariant graph generation. After constructing a forward diffusion process by an SDE to perturb graph instances towards random graphs, we design a position-enhanced graph score network to extract graph features from hybrid intermediate states, setting up the reverse-time SDE. We generate high-fidelity and diverse graphs by leveraging the numerical solvers for simulating reverse-time SDE trajectories. Experiment results show that GraphGDP can generate high-quality graphs in 24 function evaluations for efficient sampling, much faster than autoregressive models. In the future, we would like to explore this framework with lower computational complexity and further improve efficient sampling.

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China (51991395 and 62272023).

REFERENCES

[1] M. Newman, *Networks*. Oxford university press, 2018.

[2] X. Guo and L. Zhao, "A systematic survey on deep generative models for graph generation," *arXiv preprint arXiv:2007.06686*, 2020.

[3] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia, "Learning deep generative models of graphs," *arXiv preprint arXiv:1803.03324*, 2018.

[4] C. Zang and F. Wang, "Moflow: an invertible flow model for generating molecular graphs," in *SIGKDD*, 2020, pp. 617–626.

[5] C. Shi, M. Xu, Z. Zhu, W. Zhang, M. Zhang, and J. Tang, "Graphaf: a flow-based autoregressive model for molecular graph generation," in *ICLR*, 2020.

[6] Y. Luo, K. Yan, and S. Ji, "Graphdf: A discrete flow model for molecular graph generation," in *ICML*, vol. 139, 2021, pp. 7192–7203.

[7] S. Xie, A. Kirillov, R. B. Girshick, and K. He, "Exploring randomly wired neural networks for image recognition," in *ICCV*, 2019, pp. 1284–1293.

[8] B. Chen, L. Sun, and X. Han, "Sequence-to-action: End-to-end semantic graph generation for semantic parsing," *arXiv preprint arXiv:1809.00773*, 2018.

[9] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world'networks," *nature*, vol. 393, no. 6684, pp. 440–442, 1998.

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

[11] 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. 2, 2010.

[12] N. Goyal, H. V. Jain, and S. Ranu, "Graphgen: a scalable approach to domain-agnostic labeled graph generation," in *WWW*, 2020, pp. 1253–1263.

[13] 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.

[14] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.

[15] M. Simonovsky and N. Komodakis, "Graphvae: Towards generation of small graphs using variational autoencoders," in *ICANN*. Springer, 2018, pp. 412–422.

[16] Q. Liu, M. Allamanis, M. Brockschmidt, and A. Gaunt, "Constrained graph variational autoencoders for molecule design," in *NeurIPS*, 2018.

[17] A. Bojchevski, O. Shchur, D. Zügner, and S. Günnemann, "Netgan: Generating graphs via random walks," in *ICML*, 2018, pp. 610–619.

[18] N. De Cao and T. Kipf, "Molgan: An implicit generative model for small molecular graphs," *arXiv preprint arXiv:1805.11973*, 2018.

[19] J. Liu, A. Kumar, J. Ba, J. Kiros, and K. Swersky, "Graph normalizing flows," in *NeurIPS*, 2019, pp. 13 556–13 566.

[20] J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec, "Graphrnn: Generating realistic graphs with deep auto-regressive models," in *ICML*, 2018, pp. 5708–5717.

[21] R. Liao, Y. Li, Y. Song, S. Wang, W. L. Hamilton, D. Duvenaud, R. Urtasun, and R. S. Zemel, "Efficient graph generation with graph recurrent attention networks," in *NeurIPS*, 2019, pp. 4257–4267.

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

[23] X. Chen, X. Han, J. Hu, F. J. R. Ruiz, and L. Liu, "Order matters: Probabilistic modeling of node sequence for graph generation," in *ICML*, 2021, pp. 1630–1639.

[24] C. Niu, Y. Song, J. Song, S. Zhao, A. Grover, and S. Ermon, "Permutation invariant graph generation via score-based generative modeling," in *AISTATS*, 2020, pp. 4474–4484.

[25] Y. Song and S. Ermon, "Generative modeling by estimating gradients of the data distribution," in *NeurIPS*, vol. 32, 2019.

[26] S. R. De Groot and P. Mazur, *Non-equilibrium thermodynamics*. Courier Corporation, 2013.

[27] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *ICML*, 2015, pp. 2256–2265.

[28] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," in *NeurIPS*, 2020.

[29] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, "Score-based generative modeling through stochastic differential equations," in *ICLR*, 2021.

[30] L. O'Bray, M. Horn, B. Rieck, and K. Borgwardt, "Evaluation metrics for graph generative models: Problems, pitfalls, and practical solutions," in *ICLR*, 2022.

[31] R. Thompson, B. Knyazev, E. Ghalebi, J. Kim, and G. W. Taylor, "On evaluation metrics for graph generative models," in *ICLR*, 2022.

[32] B. D. Anderson, "Reverse-time diffusion equation models," *Stochastic Processes and their Applications*, vol. 12, no. 3, pp. 313–326, 1982.

[33] U. G. Haussmann and E. Pardoux, "Time reversal of diffusions," *The Annals of Probability*, pp. 1188–1205, 1986.

[34] P. Vincent, "A connection between score matching and denoising autoencoders," *Neural computation*, vol. 23, no. 7, pp. 1661–1674, 2011.

[35] S. Särkkä and A. Solin, *Applied stochastic differential equations*. Cambridge University Press, 2019, vol. 10.

[36] B. Weisfeiler and A. Leman, "The reduction of a graph to canonical form and the algebra which appears therein," *NTI, Series*, vol. 2, no. 9, pp. 12–16, 1968.

[37] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *ICLR*, 2019.

[38] C. Morris, Y. Lipman, H. Maron, B. Rieck, N. M. Kriege, M. Grohe, M. Fey, and K. Borgwardt, "Weisfeiler and leman go machine learning: The story so far," *arXiv preprint arXiv:2112.09992*, 2021.

[39] J. You, R. Ying, and J. Leskovec, "Position-aware graph neural networks," in *ICML*, 2019, pp. 7134–7143.

[40] V. P. Dwivedi and X. Bresson, "A generalization of transformer networks to graphs," *arXiv preprint arXiv:2012.09699*, 2020.

[41] P. Li, Y. Wang, H. Wang, and J. Leskovec, "Distance encoding: Design provably more powerful neural networks for graph representation learning," in *NeurIPS*, vol. 33, 2020, pp. 4465–4478.

[42] D. Kreuzer, D. Beaini, W. Hamilton, V. Létourneau, and P. Tossou, "Rethinking graph transformers with spectral attention," in *NeurIPS*, vol. 34, 2021.

[43] V. P. Dwivedi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson, "Graph neural networks with learnable structural and positional representations," *arXiv preprint arXiv:2110.07875*, 2021.

[44] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *NeurIPS*, 2017.

[45] M. M. Bronstein, J. Bruna, T. Cohen, and P. Velickovic, "Geometric deep learning: Grids, groups, graphs, geodesics, and gauges," *arXiv preprint arXiv:2104.13478*, 2021.

[46] G. Parisi, "Correlation functions and computer simulations," *Nuclear Physics B*, vol. 180, no. 3, pp. 378–384, 1981.

[47] J. Song, C. Meng, and S. Ermon, "Denoising diffusion implicit models," in *ICLR*, 2021.

[48] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, vol. 29, no. 3, pp. 93–93, 2008.

[49] I. Schomburg, A. Chang, C. Ebeling, M. Gremse, C. Heldt, G. Huhn, and D. Schomburg, "Brenda, the enzyme database: updates and major new developments," *Nucleic acids research*, vol. 32, no. suppl_1, pp. D431–D433, 2004.

[50] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola, "A kernel two-sample test," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 723–773, 2012.

[51] B. Uria, I. Murray, and H. Larochelle, "A deep and tractable density estimator," in *ICML*, 2014, pp. 467–475.

[52] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, Y. Bengio and Y. LeCun, Eds., 2015.

[53] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations," in *NeurIPS*, 2018.

[54] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *ICLR*, 2018.

[55] A. Q. Nichol and P. Dhariwal, "Improved denoising diffusion probabilistic models," in *ICML*, 2021, pp. 8162–8171.

[56] J. Jo, S. Lee, and S. J. Hwang, "Score-based generative modeling of graphs via the system of stochastic differential equations," in *ICML*, 2022, pp. 10 362–10 383.

[57] B. Srinivasan and B. Ribeiro, "On the equivalence between positional node embeddings and structural graph representations," *arXiv preprint arXiv:1910.00452*, 2019.

[58] H. Cui, Z. Lu, P. Li, and C. Yang, "On positional and structural node features for graph neural networks on non-attributed graphs," *arXiv preprint arXiv:2107.01495*, 2021.

[59] R. Sato, M. Yamada, and H. Kashima, "Approximation ratios of graph neural networks for combinatorial problems," in *NeurIPS*, vol. 32, 2019.