Autoregressive Diffusion Model for Graph Generation

Lingkai Kong[1] **Jiaming Cui**[1] **Haotian Sun**[1] **Yuchen Zhuang**[1] **B. Aditya Prakash**[1] **Chao Zhang**[1]

Abstract

Diffusion-based graph generative models have recently obtained promising results for graph generation. However, existing diffusion-based graph generative models are mostly one-shot generative models that apply Gaussian diffusion in the dequantized adjacency matrix space. Such a strategy can suffer from difficulty in model training, slow sampling speed, and incapability of incorporating constraints. We propose an *autoregressive diffusion* model for graph generation. Unlike existing methods, we define a node-absorbing diffusion process that operates directly in the discrete graph space. For forward diffusion, we design a *diffusion ordering network*, which learns a data-dependent node absorbing ordering from graph topology. For reverse generation, we design a *denoising network* that uses the reverse node ordering to efficiently reconstruct the graph by predicting the node type of the new node and its edges with previously denoised nodes at a time. Based on the permutation invariance of graph, we show that the two networks can be jointly trained by optimizing a simple lower bound of data likelihood. Our experiments on six diverse generic graph datasets and two molecule datasets show that our model achieves better or comparable generation performance with previous state-of-the-art, and meanwhile enjoys fast generation speed.

1. Introduction

Generating graphs from a target distribution is a fundamental problem in many domains such as drug discovery (Li et al., 2018), material design (Maziarka et al., 2020), social network analysis (Grover et al., 2019), and public health (Yu et al., 2020). Deep generative models have recently led to promising advances in this problem. Different from traditional random graph models (Erdos et al., 1960; Albert & Barabási, 2002), these methods fit graph data with powerful deep generative models including variational autoencoders (VAEs) (Simonovsky & Komodakis, 2018), generative adversarial networks (GANs) (Maziarka et al., 2020), normalizing flows (Madhawa et al., 2019), and energy-based models (EBMs) (Liu et al., 2021). These models are learned to capture complex graph structural patterns and then generate new high-fidelity graphs with desired properties (Zhu et al.; Du et al., 2021).

Recently, the emergence of probabilistic diffusion models has led to interest in diffusion-based graph generation (Jo et al., 2022). Diffusion models decompose the full complex transformation between noise and real data into many small steps of simple diffusion. Compared with prior deep generative models, diffusion models enjoy both flexibility in modeling architecture and tractability of the model's probability distributions. However, existing diffusion-based graph generative models (Niu et al., 2020; Jo et al., 2022; Vignac et al., 2022) suffer from three key drawbacks: (1) *Generation Efficiency*. The sampling processes are slow, as they require a very long diffusion process to arrive at the stationary noisy distribution and thus the reverse generation process is also time-consuming. (2) *Incorporating constraints*. They are all one-shot generation models and hence cannot easily incorporate constraints during the one-shot generation process. (3) *Continuous Approximation*. Niu et al. (2020); Jo et al. (2022) convert discrete graphs to continuous state spaces by adding real-valued noise to graph adjacency matrices. Such dequantization can distort the distribution of the original discrete graph structures, thus increasing the difficulty of model training.

We propose an autoregressive graph generative model named GraphArm via *autoregressive diffusion* on graphs. Autoregressive diffusion model (ARDM) (Hoogeboom et al., 2022) builds upon the recently developed absorbing diffusion (Austin et al., 2021) for discrete data, where exactly one dimension of the data decays to the absorbing state at each diffusion step. In GraphArm, we design *node-absorbing autoregressive diffusion* for graphs, which diffuses a graph directly in the discrete graph space instead of in the dequantized adjacency matrix space. The forward pass absorbs one node in each step by masking it along with its connecting edges, which is repeated until all the

[1] School of Computational Science and Engineering, Georgia Institute of Technology, Atlanta, USA. Correspondence to: Lingkai Kong <lkkong@gatech.edu>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

nodes are absorbed and the graph becomes empty. We further design a *diffusion ordering network* in GraphArm, which is jointly trained with the reverse generator to learn a data-dependent node ordering for diffusion. Compared with random ordering as in prior absorbing diffusion (Hoogeboom et al., 2022), the learned diffusion ordering not only provides a better approximation of the true marginal graph likelihood, but also eases the generative model training by leveraging structural regularities. The backward pass in GraphArm recovers the graph structure by learning to reverse the node-absorbing diffusion process with a denoising network. The reverse generative process is autoregressive, which makes GraphArm easier to handle the constraints during generation. However, a key challenge is to learn the distribution of reverse node ordering for optimizing the data likelihood. We show that this difficulty can be circumvented by just using the exact reverse node ordering and optimizing a simple lower bound of likelihood, based on the permutation invariance property of graph generation. The likelihood lower bound allows for jointly training the denoising network and the diffusion ordering network using a reinforcement learning procedure and gradient descent.

The generation speed of GraphArm is much faster than the existing graph diffusion models (Jo et al., 2022; Niu et al., 2020; Vignac et al., 2022). Due to the autoregressive diffusion process in the node space, the number of diffusion steps in GraphArm is the same as the number of nodes, which is typically much smaller than the sampling steps in (Jo et al., 2022; Niu et al., 2020; Vignac et al., 2022). Furthermore, at each step of the backward pass, we design the denoising network to predict the node type of the newly generated node and its edges with previously denoised nodes at one time. The edges to be predicted follow a mixture of multinomial distribution to ensure dependencies among each other. This makes GraphArm offer a more balanced trade-off between flexibility and efficiency.

Our key contributions are as follows: (1) To the best of our knowledge, our work is the first *autoregressive* diffusion-based graph generation model, underpinned by a new node self-absorbing diffusion process. Our model represents a generalized form of the diffusion processes, a class in which ARDM (Hoogeboom et al., 2022) and diffusion Schrödinger bridge (De Bortoli et al., 2021) also fall. (2) GraphArm learns a data-dependent node generation ordering and thus better leverages the structural regularities for autoregressive graph diffusion. (3) We validate our method on eight graph generation tasks, on which we show that GraphArm outperforms existing graph generative models and is efficient in generation speed.

2. Additional Related Work

Graph Generation. *One-shot graph generative models* generate all edges between nodes at once. Models based on VAEs (Simonovsky & Komodakis, 2018; Liu et al., 2018; Ma et al., 2018) and GANs (Cao & Kipf, 2022; Maziarka et al., 2020) generate all edges independently from latent embeddings. This independence assumption can hurt the quality of the generated graphs. Normalizing flow models (Zang & Wang, 2020; Madhawa et al., 2019) are restricted to invertible model architectures for building a normalized probability. The other class is *autoregressive graph generative models*, which generate a graph by sequentially adding nodes and edges. Autoregressive generation can be achieved using recurrent networks (Li et al., 2018; You et al., 2018b; Dai et al., 2020), VAEs (Liu et al., 2018; Jin et al., 2018; 2020; Guo et al., 2021), normalizing flows (Shi et al., 2020; Luo et al., 2021), and Reinforcement Learning (RL) (You et al., 2018a). By breaking the problem into smaller parts, these methods are more apt at capturing complex structural patterns and can easily incorporate constraints during generation. However, a key drawback of them is that their training is sensitive to node ordering. Most existing works pre-define a fixed node ordering by ad-hoc such as breadth-first search (BFS) ordering (You et al., 2018b; Shi et al., 2020), which can be suboptimal for generation. OM (Chen et al., 2021) also learns the node ordering for autoregressive graph generation, but our method differs from OM in two aspects. (1) The motivations are different. Our method is motivated by autoregressive diffusion and treats the graph sequences in the diffusion process as the latent variable; OM treats the node ordering as the latent variable and infers its posterior distribution in a way similar to Variational autoencoder (VAE). (2) Our training objective is much simpler than OM. First, we do not need to compute the complicated graph automorphism, which requires some approximation algorithms to compute. Second, our training objective does not involve the entropy of the node ordering distribution. Existing works (Lucas et al., 2019b;a) have shown that the VAE objective can cause posterior collapse.

Diffusion and Score-Based Generation. Diffusion models have emerged as a new family of powerful deep generative models. Denoising diffusion probabilistic modeling (DDPM) (Sohl-Dickstein et al., 2015; Ho et al., 2020) perturbs the data distribution into a Gaussian distribution through an forward Markov noising process, and then learns to recover data distribution via the reverse transition of the Markov chain. Closely related to DDPM is score-based generation (Song & Ermon, 2019), which perturbs data with gradually increasing noise, and then learns to reverse the perturbation via score matching. Song et al. (2021) generalize diffusion models to continuous-time diffusion using forward and backward SDEs. Existing diffusion-based graph generation models are all one-shot. Niu et al. (2020) model the adjacency matrices using score matching at different noise scales, and uses annealed Langevin dynamics for generation; Jo et al. (2022) propose a continuous-time graph diffusion model that jointly models adjacency matrices and node features through stochastic differential equations (SDEs). Recently, Vignac et al. (2022) introduce a discrete graph diffusion process by defining a Markov transition matrix for different node and edge types while Haefeli et al. (2022) propose a similar model but only for the non-attributed graph. Different from these works, our model is the first *autoregressive* diffusion model for graph generation, which defines diffusion directly in the discrete graph space. As we point out in Section 1, our framework represents a generalized form of diffusion models, encompassing a broader class of models, including ARDM (Hoogeboom et al., 2022) and diffusion Schrödinger bridge (De Bortoli et al., 2021).

3. Background

Diffusion Model and Absorbing Diffusion Given a training instance $\boldsymbol{x}_0 \in \mathbb{R}^D$ sampled from the underlying distribution $p_{\text{data}}(\boldsymbol{x}_0)$, a diffusion model defines a forward Markov transition kernel $q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$ to gradually corrupt training data until the data distribution is transformed into a simple noisy distribution. The model then learns to reverse this process by learning a denoising transition kernel parameterized by a neural network $p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$.

Most existing works on diffusion models use Gaussian diffusion for continuous-state data. To apply Gaussian diffusion on discrete data, one can use the dequantization method by adding small noise to the data. However, dequantization distorts the original discrete distribution, which can cause difficulty in training diffusion-based models. For example, dequantization on graph adjacency matrices can destroy graph connectivity information and hurt message passing. (Austin et al., 2021; Hoogeboom et al., 2021) introduce several discrete state space diffusion models using different Markov transition matrices. Among them, absorbing diffusion is a promising one due to its simplicity and strong empirical performance.

Definition 1 (Absorbing Discrete Diffusion)**.** *An absorbing diffusion is a Markov destruction process defined in the discrete state space. At transition time step t, each element $\boldsymbol{x}_t^{(i)}$ in dimension i is independently decayed into an absorbing state with probabilities $\alpha(t)$.*

The absorbing state can be a [MASK] token for texts or gray pixel for images (Austin et al., 2021). The diffusion process will converge to a stationary distribution that has all the mass on the absorbing state. The reverse of the absorbing diffusion is learned with categorical distribution to generate the original data. Typically, the decaying probabilities $\alpha(t)$ need to be small and the diffusion steps T need to be large to attain good performance.

Autoregressive Diffusion Model Although absorbing diffusion directly operates on the discrete data, it still needs a large number of diffusion steps and thus its generation process can be slow. Autoregressive diffusion model (ARDM) (Hoogeboom et al., 2022) describes a fixed absorbing diffusion process where the generative steps is equal to the dimensionality of the data, *e.g.*, the number of pixels in an image or the number of tokens in a piece of text.

Definition 2 (Autoregressive Diffusion)**.** *An autoregressive diffusion is a stochastic absorbing process where exactly one dimension decays to the absorbing state at a time. The diffusion is repeated until all dimensions are absorbed.*

An equivalent way to describe this process is to first sample a permutation $\sigma \in S_D$, where S_D represents the set of all permutations of the dimension indices $1, \cdots, D$. Then each dimension of the data decays in that order towards the absorbing state. The corresponding generative process then models the variables in the *exact opposite order* of the permutation. ARDM amounts to absorbing diffusion with a continuous time limit, as detailed in Appendix A.3.

While ARDM offers an efficient and general diffusion framework for discrete data, two key questions remain to be addressed for applying ARDM for graphs: (1) How do we define absorbing states for inter-dependent nodes and edges in graphs without losing the efficiency of ARDM? (2) While ARDM imposes a uniform ordering for arriving at an order-agnostic variational lower bound (VLB) of likelihood, a random ordering fails to capture graph topology. How do we obtain a data-dependent ordering that leverages graph structural regularities during generation? In the next section, we address these two challenges in our proposed model.

4. Method

A graph is represented by the tuple $G = (V, E)$ with node set $V = \{v_1, \cdots, v_n\}$ and edge set $E = \{e_{v_i,v_j}|v_i, v_j \in V)\}$. We denote by $n = |V|$ and $m = |E|$ the number of nodes and edges in G respectively. Each node and edge have their corresponding categorical labels, *e.g.*, $e_{v_i,v_j} = k$ represents that the edge between node v_i and v_j is in type k. We treat the absence of edges between two nodes as a particular edge type. Our goal is to learn a graph generative model from a set of training graphs.

4.1. Autoregressive Graph Diffusion Process

Due to the dependency between nodes and edges, it is non-trivial to apply absorbing diffusion (Austin et al., 2021) in the discrete graph space. We first define absorbing node state on graphs as follows:

Definition 3 (Absorbing Node State)**.** *When a node v_i enters the absorbing state, (1) it will be masked and (2) it will be connected to all the other nodes in G by masked edges.*

Instead of only masking the original edges, we connect the masked node v_i to all the other nodes with masked edges

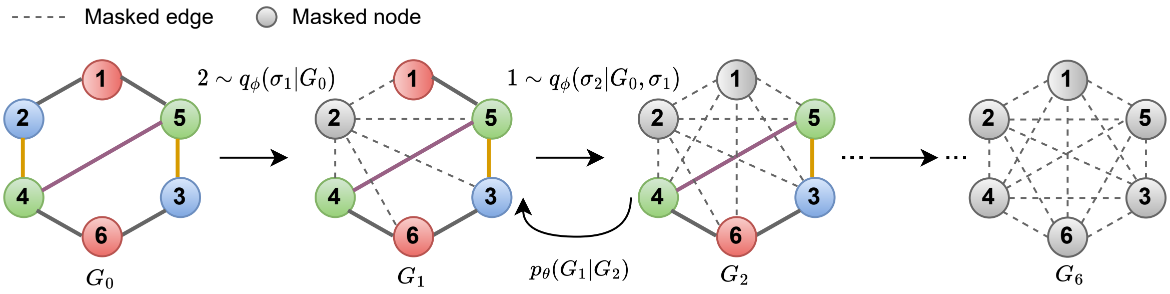
Masked edge
Masked node
$2 \sim q_\phi(\sigma_1|G_0)$
$1 \sim q_\phi(\sigma_2|G_0, \sigma_1)$
$p_\theta(G_1|G_2)$
G_0
G_1
G_2
G_6

Figure 1. The autoregressive graph diffusion process. In the forward pass, the nodes are autoregressively decayed into the absorbing states, dictated by an ordering generated by the diffusion ordering network $q_\phi(\sigma|G_0)$. In the reverse pass, the generator network $p_\theta(G_t|G_{t+1})$ reconstructs the graph structure using the reverse node ordering. Note that we do not need to consider the graph automorphism as (Chen et al., 2021), since the diffusion process assigns a unique ID to each node in G_0 to obtain the decay ordering. Therefore, there is a one-to-one mapping between $G_{0:n}$ and $\sigma_{1:n}$. For example, v_1 and v_6 have the same topology, but the denoising network will recover the exact node v_1 at $t = 2$ since $\sigma_2 = 1$. We provide more illustrations in Appendix. A.5

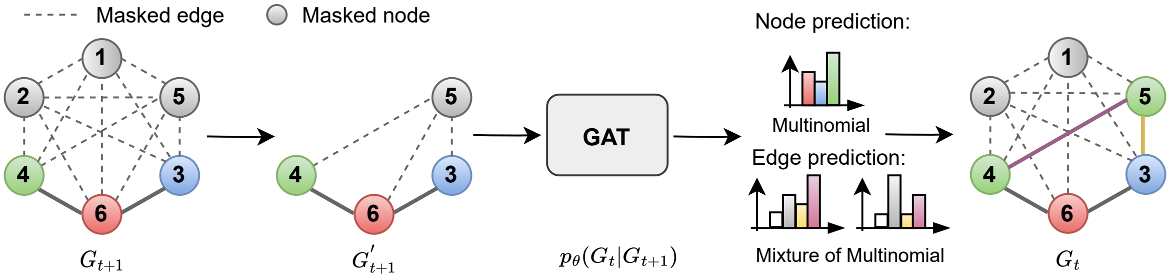

Figure 2. The generation procedure at step t with the denoising network $p_\theta(G_t|G_{t+1})$. The denoising network predicts the node type of node v_{σ_t} and its edges with all previously denoised nodes.

as we cannot know v_i's original neighbors in the absorbing state. With the absorbing node state defined, we then need a node decay ordering for the forward absorbing pass. A naïve strategy is to use a random ordering sampled from a uniform distribution as in (Hoogeboom et al., 2022). In the reverse generation process, the variables will be generated in the exact reverse order, which also follows a uniform distribution. However, such a strategy is problematic for graphs. First, different graph datasets have different structural regularities, and it is key to leverage such regularities to ease generative learning. For example, community-structured graphs typically consist of dense subgraphs that are loosely overlapping. For such graphs, it is an easier learning task to generate one community first and then add the others, but a random node ordering cannot leverage such local structural regularity, which makes generation more difficult. Second, to compute the likelihood, we need to marginalize over all possible node orderings due to node permutation invariance. It will be more sample efficient if we can use an optimized proposal ordering distribution and use importance sampling to compute the data likelihood.

To address this issue, we propose to use a diffusion ordering network $q_\phi(\sigma|G_0)$ such that, at each diffusion step t, we sample from this network to select a node $v_{\sigma(t)}$ to be absorbed and obtain the corresponding masked graph G_t (Figure 1). This leads to the following definition of our graph autoregressive diffusion process:

Definition 4 (Autoregressive Graph Diffusion Process)**.** *In autoregressive graph diffusion, the node decay ordering σ is sampled from a diffusion ordering network $q_\phi(\sigma|G_0)$. Then, exactly one node decays to the absorbing state at a time according to the sampled diffusion ordering. The process proceeds until all the nodes are absorbed.*

The diffusion ordering network follows a recurrent structure $q_\phi(\sigma|G_0) = \prod_t q_\phi(\sigma_t|G_0, \sigma_{(<t)})$. At each step t, the distribution of the t-th node σ_t is conditioned on the original graph G_0 and the generated node ordering up to $t-1$, *i.e.*,$\sigma_{(<t)}$. We use a graph neural network (GNN) to encode the structural information in the graph. To capture the partial ordering, we add positional encodings into node features (Vaswani et al., 2017) as in (Chen et al., 2021). We denote the updated node embedding of node v_i after passing the GNN as $\boldsymbol{h}_i^d$, and parameterize $q_\phi(\sigma_t|G_0, \sigma_{(<t)})$ as a categorical distribution:

$$q_\phi(\sigma_t|G_0, \sigma_{(<t)}) = \frac{\exp(\boldsymbol{h}_i^d)}{\sum_{i' \notin \sigma_{(<t)}} \exp(\boldsymbol{h}_{i'}^d)}. \qquad (1)$$

With $q_\phi(\sigma|G_0)$, GraphArm can learn to optimize node ordering for diffusion. However, this also requires us to infer the reverse generation ordering in the backward pass. Inferring such a reverse generation ordering is difficult since we do not have access to the original graph G_0 in intermediate backward steps. In Section 4.3, we show that it is possible to circumvent inferring this generation ordering by leveraging the permutation invariance of graph generation.

4.2. The Reverse Generative Process

In the generative process, a denoising network $p_\theta(G_t|G_{t+1})$ will denoise the masked graph in the reverse order of the diffusion process. We design $p_\theta(G_t|G_{t+1})$ as a graph attention network (GAT) (Veličković et al., 2018; Liao et al., 2019) parameterized by θ, so that the model can distinguish the masked and unmasked edges. For clarity, we use the Vanilla GAT to illustrate the computing process. However, one can adopt any advanced graph neural network with attentive message passing.

At time t, the input to the denoising network $p_\theta(G_t|G_{t+1})$ is the previous masked graph G_{t+1}. A direct way is to use G_{t+1} which contains all the masked nodes with their corresponding masked edges. However, during the initial generation steps, the graph is nearly fully connected with masked edges. This has two issues: (1) the message passing procedure will be dominated by the masked edges which makes the messages uninformative. (2) Storing the dense adjacency matrix is memory expensive, which makes the model unscalable to large graphs. Therefore, during each generation step, we only keep the masked node to be denoised with its associated masked edges, while ignoring the other masked nodes. We refer the modified masked graph as $G_t^{'}$, as shown in Figure 2.

The denoising network first uses an embedding layer to encode each node v_i into a continuous embedding space, *i.e.*, $\boldsymbol{h}_i = \text{Embedding}(v_i)$. At l-th message passing, we update the embedding of node v_i by aggregating the attentive messages from its neighbor nodes: $\alpha_{i,j} = \frac{\exp(\text{LeakyReLU}(\boldsymbol{a}^T[\boldsymbol{W}\boldsymbol{h}_i||\boldsymbol{W}\boldsymbol{h}_j])}{\sum_{k\in\mathcal{N}_i}\exp(\text{LeakyReLU}(\boldsymbol{a}^T)[\boldsymbol{W}\boldsymbol{h}_i||\boldsymbol{W}\boldsymbol{h}_j])}, \boldsymbol{h}_i = \text{ReLU}\left(\sum_{j\in\mathcal{N}_i}\alpha_{i,j}\boldsymbol{W}\boldsymbol{h}_j\right)$, where $\boldsymbol{W}$ is the weight matrix, $\boldsymbol{a}$ is the attention vector. The attention mechanism enables the model to distinguish if the message comes from a masked edge. After L rounds of message passing, we obtain the final embedding $\boldsymbol{h}_i^L$ for each node, then we predict the node type of the new node v_{σ_t} and the edge types between v_{σ_t} and all previously denoised nodes $\{v_{\sigma(>t)}\}$. The node type prediction follows a multinomial distribution. For edge prediction, one choice is to sequentially predict these edges as in (You et al., 2018b; Shi et al., 2020). However, this sequential generation process is inefficient and takes $\mathcal{O}(n^2)$. Instead, we predict the connections of the new node to all previous nodes at once using a mixture of multinomial distribution. The mixture distribution can capture the dependencies among edges to be generated and meanwhile reduce the autoregressive generation steps to $\mathcal{O}(n)$.

4.3. Training Objective

We use approximate maximum likelihood as the training objective for GRAPHARM. We first derive the variational lower bound (VLB) of likelihood as:

$$\begin{aligned}\log p_\theta(G_0) &= \log\left(\int p(G_{0:n})\frac{q(G_{1:n}|G_0)}{q(G_{1:n}|G_0)}dG_{1:n}\right)\\ &\geq \mathbb{E}_{q(\sigma_{1:n}|G_0)}\sum_t \log p_\theta(G_t|G_{t+1})\\ &\quad - \text{KL}(q_\phi(\sigma_{1:n}|G_0)|p_{\theta'}(\sigma_{1:n}|G_n)), \end{aligned} \tag{2}$$

where $G_{0:n}$ denotes all values of G_t for $t = 0, \cdots, n$ and $p_{\theta'}(\sigma_{1:n}|G_n)$ is the distribution of the generation ordering. A detailed derivation of Eq. 2 is given in Appendix A.4

As we can see from Eq. 2, the diffusion process introduces a separate reverse generation ordering network $p_{\theta'}(\sigma_{1:n}|G_n)$. Learning $p_{\theta'}(\sigma_{1:n}|G_n)$ is nontrivial as we do not have the original graph G_0 in the intermediate generation process. However, we show that we can avoid such difficulty and simply ignore the KL-divergence term. While the generation ordering network is required for non-graph data such as text to determine which token to unmask at test time, it is not needed for graph generation due to node permutation invariance. The first term will encourage the denoising network $p_\theta(G_t|G_{t+1})$ to predict the node and edge types in the exact reverse ordering of the diffusion process, thus the denoising network itself can be a proxy of the generation ordering. Due to permutation invariance, we can simply replace any masked node and its masked edges with the predicted node and edge types at each time step. Therefore, we can ignore the second term and finally arrive at a simple training objective:

$$\begin{aligned}L_{\text{train}} &= \mathbb{E}_{\sigma_{1:n}\sim q_\phi(\sigma_{1:n}|G_0)}\sum_t p_\theta(G_t|G_{t+1}) \qquad (3)\\ &= n\mathbb{E}_{\sigma_{1:n}\sim q_\phi(\sigma_{1:n}|G_0)}\mathbb{E}_{t\sim\mathcal{U}_n}p_\theta(O_{v_{\sigma_t}}^{\sigma(>t)}|G_{t+1}),\end{aligned}$$

where the last equivalence comes from treating t as the random variable with a uniform distribution $\mathcal{U}_n$ over 1 to n. $O_{v_{\sigma_t}}^{\sigma(>t)}$ represents the node type of v_{σ_t} and its edges with all previously denoised nodes, *i.e.*, $\{v_{\sigma_t}, \{e_{v_{\sigma_t},v_j}\}_{j=\sigma_{t+1}}^{\sigma_n}\}$.

Compared with the random diffusion ordering, our design has two benefits: (1) We can automatically learn a data-dependent node generation ordering which leverages the graph structural information. (2) We can consider the diffusion ordering network as an optimized proposal distribution of importance sampling for computing the data likelihood, which is more sample-efficient than a uniform proposal distribution.

Soft Label Training The architecture of ARDM (Hoogeboom et al., 2022) predicts all masked dimensions simultaneously, which enables training the univariate conditionals $p(\boldsymbol{x}_k|\boldsymbol{x}_{\sigma(>t)})$ for all $k \in \sigma_{(\leq t)}$ in parallel. In GraphArm, due to the node permutation invariant property of graph, this parallel training can be simplified as training with a soft label which is weighted by the probability given by the diffusion ordering network:

$$L_{\text{train}} = n\mathbb{E}_{q_\phi(\sigma_{1:n}|G_0)}\mathbb{E}_{t\sim\mathcal{U}_n} \sum_{k\in\sigma_{(\leq t)}} w_k p_\theta(O_{v_k}^{\sigma_{(>t)}}|G_{t+1}),$$

where $w_k = q_\phi(\sigma_t = k|G_0, \sigma_{(<t)})$. In practice, the probability mass $q_\phi(\sigma_t|G_0, \sigma_{(<t)})$ might be concentrated around a small set of remaining nodes. Therefore, it is generally sufficient to consider only those node labels associated with the highest probabilities.

4.4. Parameter Optimization

Learning the parameters of GraphArm is challenging, because we need to evaluate the expectation of the likelihood over the diffusion ordering network. We use a reinforcement learning (RL) procedure by sampling multiple diffusion trajectories, thereby enabling training both the diffusion ordering network $q_\phi(\sigma|G_0)$ and the denoising network $p_\theta(G_t|G_{t+1})$ using gradient descent.

Specifically, at each training iteration, we explore the diffusion ordering network by creating M diffusion trajectories for each training graph $G_0^{(i)}$. Each trajectory is a sequence of graphs $\{G_t^{i,m}\}_{1\leq t\leq n}$ where the node decay ordering $\sigma^{i,m}$ is sampled from $q_\phi(\sigma|G_0^{i,m})$. For each trajectory, we sample T time steps. The denoising network $p_\theta(G_t|G_{t+1})$ is then trained to minimize the negative VLB using stochastic gradient descent (SGD):

$$\triangle\theta \leftarrow \frac{\eta_1}{M}\nabla \sum_{i\in\mathcal{B}_{\text{train}}} \sum_{m,t} \sum_{k\in\sigma_{(\leq t)}} \frac{n_i w_k^{i,m}}{T} \log p_\theta(O_{v_k}^{\sigma_{(>t)}}|G_{t+1}^{i,m}),$$

where $\mathcal{B}_{\text{train}}$ is the a minibatch sampled from the training data and $w_k^{i,m} = q_\phi(\sigma_t^{i,m} = k|G_0^i, \sigma_{(<t)}^{i,m})$.

To evaluate the current diffusion ordering network, we create M trajectories for each validation graph and compute the negative VLB of the denoising network to obtain the corresponding rewards $R^{i,m} = -\sum_t \sum_{k\in\sigma_{(\leq t)}} \frac{n_i}{T} w_k^{i,m} \text{log} p_\theta(O_{v_k}^{\sigma_{(>t)}}|G_{t+1}^{i,m})$. Then, the diffusion ordering network can be updated with common RL optimization methods, *e.g.*, the REINFORCE algorithm (Williams, 1992):

$$\triangle\phi \leftarrow \frac{\eta_2}{M} \sum_{i\in\mathcal{B}_{\text{val}}} \sum_{m} R^{i,m}\nabla \log q_\phi(\sigma|G_0^{i,m}). \quad (4)$$

The detailed training procedure is summarized in Algorithm 1 in Appendix. A.6.

5. Experiments

5.1. Generic Graph Generation

Experimental Setup. We evaluate the performance of GraphArm on six diverse graph generation benchmarks from different domains: (1) Community-small (You et al., 2018b), (2) Caveman (You, 2018), (3) Cora (Sen et al., 2008), (4) Breast (Gonzalez-Malerva et al., 2011), (5) Enzymes (Schomburg et al., 2004) and (6) Ego-small (Sen et al., 2008). For each dataset, we use 80% of the graphs as training set and the rest 20% as test sets. Following (Liao et al., 2019), we randomly select 20% from the training data as the validation set. We generate the same amount of samples as the test set for each dataset. More details can be seen in Appendix.A.10. Following previous work (You et al., 2018b), we measure generation quality using the maximum mean discrepancy (MMD) as a distribution distance between the generated graphs and the test graphs. Specifically, we compute the MMD of degree distribution, clustering coefficient, and orbit occurrence numbers of 4 nodes between the generated set and the test set. We also report the generation time of different methods.

Baselines and Implementation Details. We compare GraphArm with the following baselines: DeepGMG (Li et al., 2018) and GraphRNN (You et al., 2018b) are RNN-based autoregressive graph generation models. GraphAF (Shi et al., 2020) and GraphDF (Luo et al., 2021) are flow-based autoregressive models. GRAN (Liao et al., 2019) is an autoregressive model conditioned on blocks. OM (Chen et al., 2021) is an autoregressive model that infers node ordering using variational inference. GraphVAE (Simonovsky & Komodakis, 2018) is an one-shot model based on VAE. SPECTRE (Martinkus et al., 2022) is a one-shot model based on GAN. DiGress (Vignac et al., 2022), EDP-GNN (Niu et al., 2020) and GDSS (Jo et al., 2022) are diffusion-based one-shot methods. Implementation details and parameter settings are provided in in Appendix A.7.

Experimental Results Table 1 shows the generation performance on the six benchmarks for all the methods. (1) As we can see, our method can outperform or achieve competitive performance compared with the baselines. In terms of efficiency, our model is on par with the most efficient autoregressive baseline GRAN, and 10-100X faster than other baselines except for the one-shot baselines GraphVAE and SPECTRE. However, GraphVAE's generation quality is much worse than ours while SPECRE is a GAN based model and cannot provide likelihood computation. (2) GDSS and DiGress are the strongest baselines. However, their generation speeds are extremely slow as they need a long diffusion process to arrive at the stationary distribution. Our GraphArm is up to 100X times faster than GDSS and 30X times faster than DiGress. The other graph diffusion model EDP-GNN is even slower as its generation process uses

Model	Community-small				Caveman				Cora			
	Deg.	**Clus.**	**Orbit**	**Time/s**	**Deg.**	**Clus.**	**Orbit**	**Time/s**	**Deg.**	**Clus.**	**Orbit**	**Time/s**
DeepGMG	0.220	0.950	0.400	496.6	1.752	1.642	0.2122	530.2	-	-	-	-
GraphRNN	0.080	0.120	0.040	16.4	0.371	1.035	0.033	27.0	1.689	0.608	0.308	33.3
GraphAF	0.180	0.200	0.020	19.3	0.269	0.587	0.422	20.5	0.176	0.080	0.094	108.5
GraphDF	0.060	0.120	0.030	10.3	0.077	0.373	0.051	18.5	0.454	**0.074**	0.256	129.7
GraphVAE	0.350	0.980	0.540	0.2	1.402	1.086	1.391	0.2	1.521	1.740	0.788	0.3
GRAN	0.060	0.110	0.050	1.8	0.043	0.130	0.018	2.5	0.125	0.272	0.127	5.1
OM	0.047	0.130	0.008	2.0	0.032	0.076	0.027	3.0	0.249	0.201	0.145	5.6
EDP-GNN	0.053	0.144	0.026	$2.9e^3$	0.032	0.168	0.030	$1.8e^3$	**0.093**	0.269	0.062	$4.6e^3$
SPECTRE	0.048	0.049	0.016	0.4	**0.013**	0.084	0.028	0.5	0.021	0.080	**0.007**	0.14
GDSS	0.045	0.086	0.007	$4.3e^2$	0.019	0.048	**0.006**	$5.1e^2$	0.160	0.376	0.187	$4.2e^2$
DiGress	0.047	**0.041**	0.026	5.7	0.019	0.040	**0.003**	10.4	0.044	0.042	0.223	79.7
GraphArm	**0.034**	0.082	**0.004**	1.9	0.039	**0.028**	0.018	2.7	0.273	0.138	0.105	4.5

Model	Breast				Enzymes				Ego-small			
	Deg.	**Clus.**	**Orbit**	**Time/s**	**Deg.**	**Clus.**	**Orbit**	**Time/s**	**Deg.**	**Clus.**	**Orbit**	**Time/s**
DeepGMG	-	-	-	-	-	-	-	-	0.040	0.100	0.020	477
GraphRNN	0.103	0.138	0.005	31.0	0.017	0.062	0.046	19.2	0.090	0.220	0.003	18.7
GraphAF	0.111	0.407	0.003	53.1	1.669	1.283	0.266	28.6	0.030	0.110	**0.001**	6.9
GraphDF	0.283	0.078	0.035	62.4	1.503	1.061	0.202	39.8	0.040	0.130	0.010	10.2
GraphVAE	1.591	1.993	1.050	0.2	1.369	0.629	0.191	0.7	0.130	0.170	0.050	0.3
GRAN	0.073	0.413	0.010	2.1	0.054	0.078	0.017	3.2	0.030	0.029	0.014	1.2
OM	0.042	0.140	0.005	2.8	0.051	0.083	0.024	2.4	0.024	0.035	0.018	1.4
EDP-GNN	0.131	0.038	0.019	$3.6e^3$	0.023	0.268	0.082	$2.2e^3$	0.052	0.093	0.007	$3.9e^3$
SPECTRE	0.312	0.837	0.087	0.14	0.136	0.195	0.125	0.9	0.078	0.078	0.007	0.4
GDSS	0.113	**0.020**	0.003	$8.6e^2$	0.026	0.061	0.009	$4.4e^2$	0.021	0.024	0.007	$2.6e^2$
DiGress	0.152	0.024	0.008	91.5	**0.004**	0.083	**0.002**	89.48	**0.015**	0.029	0.005	8.8
GraphArm	**0.036**	0.041	**0.002**	2.3	0.029	**0.054**	0.015	3.5	0.019	**0.017**	0.010	1.2

Table 1. Generation results on the six generic graph datasets for all the methods. Best results are bold and the second best values are underlined (smaller the better). "-" denotes out-of-resources that take more than 10 days to run.

annealed Langevin Dynamics with many different noise levels. (3) GraphArm also outperforms existing autoregressive graph generative models by large margins. This is because they adopt a fixed node ordering when training the generative model while GraphArm automatically learns a data-dependent generation ordering for the graph. Though OM also learns a node ordering distribution, GraphArm consistently outperforms it. This is because OM uses the VAE objective which may suffer from posterior collapse, while GraphArm has a much simpler training objective based on autoregressive diffusion.

5.2. Molecule Generation

Experimental Setup We use two molecular dataset, QM9 (Ramakrishnan et al., 2014) and ZINC250k (Irwin et al., 2012). Following previous works (Jo et al., 2022), we evaluate 10,000 generated molecules with the following metrics. **Fréchet ChemNet Distance (FCD)** (Preuer et al., 2018) evaluates the distance between the training and generated sets using the activations of the penultimate layer of the ChemNet. **Neighborhood subgraph pairwise distance kernel (NSPDK) MMD** (Costa & De Grave, 2010) is the MMD between the generated molecules and test molecules which takes into account both the node and edge features for evaluation. **Validity** is the fraction of valid molecules without valency correction. **Uniqueness** is the fraction of the valid molecules that are unique. **Novelty** is the fraction of the valid molecules that are not included in the training set. We provide the implementation details in Appendix. A.8.

Experimental Results Table. 2 shows the generation performance on the two molecule datasets for all the methods. GraphArm can achieve competitive performance with the strongest baseline while being much more efficient for generation. GraphArm is performing well, particularly on NSPDK and FCD which are two salient metrics measuring how close the generated molecules lie to the distribution in graph structure space and chemical space. On QM9, only DiGress outperforms GraphArm. However, its novelty score is only below 40%. This indicates that DiGress may suffer from simply memorizing the training data. On ZINC250K, only GDSS outperforms GraphArm but 10X slower for generation. In terms of efficiency, only MoFlow and SPECTRE are faster than GraphArm but their generation performance is not on par with GraphArm. For the validity, though GraphArm is not the best one, this can be easily addressed using edge resampling during generation since GraphArm is an autoregressive model which is more

	QM9						ZINC250k					
Model	Validity↑	NSPDK↓	FCD↓	Unique↑	Novelty↑	Time↓	Validity↑	NSPDK↓	FCD↓	Unique↑	Novelty↑	Time↓
GraphAF	74.43	0.020	5.27	88.64	86.59	3.01e^3	68.47	0.044	16.02	98.64	100	7.2e^3
GraphDF	93.88	0.064	10.93	98.58	98.54	5.8e^4	90.61	0.177	33.55	99.63	100	6.95e^4
MoFlow	91.36	0.017	4.47	98.65	94.72	5.43	63.11	0.046	20.93	99.99	100	36.1
EDP-GNN	47.52	0.005	2.68	99.25	86.58	52.e^2	82.97	0.049	16.74	99.79	100	1.21e^4
GraphEBM	8.22	0.030	6.14	97.90	97.01	44.2	5.29	0.212	35.47	98.79	100	87.2
SPECTRE	87.3	0.163	47.96	35.7	97.28	3.3	90.2	0.109	18.44	67.05	100	103.1
GDSS	95.72	0.003	2.9	98.46	86.27	1.3e^2	**97.01**	**0.019**	**14.66**	99.64	100	2.58e^3
DiGress	**99.0**	**0.0005**	**0.36**	96.66	33.4	1.02e^2	91.02	0.082	23.06	81.23	100	1.52e^3
GraphArm	90.25	0.002	1.22	95.62	70.39	15.2	88.23	0.055	16.26	99.46	100	132.8

Table 2. Generation results on the QM9 and ZINC250k dataset. Best results are bold and the second best values are underlined.

flexible. The outstanding performance of GraphArm on molecule datasets verifies the effectiveness of our method for learning the underlying distribution of graphs with multiple node and edge types.

5.3. Constrained Graph Generation

In this set of experiments, we study GraphArm's capability in constrained graph generation by comparing it with the strongest one-shot and autoregressive baselines. We use the Caveman dataset for constrained graph generation, with the constraint that the maximum node degree is no larger than 6. The detailed setup is in Appendix A.9. Table. 3 shows the constrained generation performance on the Caveman dataset. We find that more than half of their generated samples are invalid. For autoregressive baselines, we apply the degree-checking procedure on the two strongest baselines, *i.e.*,GRAN and OM. DiGress, GDSS and EDP-GNN are all one-shot generative models and they are hard to incorporate such constraints during the generation procedure. Though it is possible to apply a post-hoc modification for DiGress, GDSS and EDP-GNN, such a strategy has not been explored in the existing literature for general graph generation. As we can see, GraphArm can generate constrained samples that are closer to the data distribution. This is useful for many real-world applications. For example, when designing the contact networks of patients and healthcare workers in hospitals, a constraint of degrees for healthcare workers can help avoid superspreaders and potential infectious disease outbreaks (Jang et al., 2019; Adhikari et al., 2019).

Method	Validity	Deg.	Clus.	Orbit
EDP-GNN	39%	-	-	-
GDSS	34%	-	-	-
DiGress	33%	-	-	-
GRAN	100%	0.208	0.231	0.158
OM	100%	0.190	0.168	**0.132**
GraphArm	100%	**0.176**	**0.157**	0.144

Table 3. Constrained graph generation results on the Caveman dataset.

Method	Deg.	Clus.	Orbit	Counts
OA-ARDM	0.085	0.129	0.032	5.6
GraphArm	**0.034**	**0.082**	**0.004**	**2.6**

Table 4. Generation performance and node generation ordering of GraphArm and OA-ARDM on the Community-small dataset. Counts represents the average number of nodes that cross different clusters during the generation procedure.

5.4. Ablation Study: Effect of Diffusion Ordering

We further validate the superiority of the learned diffusion ordering by comparing GraphArm with an ablation OA-ARDM, which uses random node ordering (Hoogeboom et al., 2022) for graph generation. Table 4 shows the generation performance and generation ordering of GraphArm and OA-ARDM on the Community-small dataset. As shown, with the random generation ordering, the generation performance drops significantly. To evaluate the node generation ordering, we use the spectral graph clustering method to partition the nodes into two clusters for each generated graph and then count the cross-cluster steps during the generation procedure. As we can see, the average number of nodes that cross different clusters of GraphArm is much smaller than OA-ARDM. This demonstrates that GraphArm tends first to generate one cluster and then adds another cluster while OA-ARDM just randomly generates the graph. Therefore, the generation ordering of GraphArm can better capture graph topology regularity than OA-ARDM. Figure 3 (in Appendix A.1) visualizes the graph generation process. As we can see, GraphArm first generates one community and then moves to another; while OA-ARDM randomly generates the graph and fails to capture the underlying graph distribution.

We remark that the learned node orderings can only be transferred when the two datasets share structural similarities, as our learned node ordering is data-dependent. For instance, transferability is possible if both datasets represent social communities with similar structural properties, e.g., several dense subgraphs that are loosely overlapping. However, if the datasets do not exhibit such similarities, the node ordering cannot be effectively transferred.

6. Limitations and Discussion

We have proposed a new autoregressive diffusion model for graph generation. The proposed model GRAPHARM defines a node-absorbing diffusion process that directly operates in the discrete graph spaces. We designed a diffusion ordering network that learns a data-dependent ordering for this diffusion process, coupled with a reverse denoising network that performs autoregressive graph reconstruction. We derived a simple variational lower bound of the likelihood, and showed that the two networks can be jointly trained with reinforcement learning. Our experiments have validated the generation performance and efficiency of GRAPHARM.

We discuss limitations and possible extensions of GRAPHARM: (1) *Handling more complex constraints.* We have shown that the autoregressive procedure of GRAPHARM can handle constraints better than one-shot generative models. However, practical graph generation applications can involve complex constraints on global-level properties such as number of articulation points, which is challenging for our model and existing graph generative models. (2) *Memory and time efficiency.* We have shown that our model is much faster than previous one-shot diffusion-based graph generation. There are efforts on accelerating the sampling process of diffusion models (De Bortoli et al., 2021). It remains an open problem to investigate how effective can such techniques accelerate diffusion-based graph generation without compromising generation performance. In addition, unlike our GRAPHARM model, these one-shot diffusion models need to maintain the full adjacency matrix in memory. This results in large memory footprints and can make them difficult to handle large graphs.

Acknowledgements

We thank the anonymous reviewers for their helpful comments. This work was supported in part by the NSF (Expeditions CCF-1918770, CAREER IIS-2028586, IIS-2027862, IIS-1955883, IIS-2106961, IIS-2008334, CAREER IIS-2144338, PIPP CCF-2200269), CDC MInD program, faculty research award from Facebook and funds/computing resources from Georgia Tech.

References

Caveman dataset. https://github.com/JiaxuanYou/graph-generation/blob/master/create_graphs.py, 2018.

Adhikari, B., Lewis, B., Vullikanti, A., Jiménez, J. M., and Prakash, B. A. Fast and near-optimal monitoring for healthcare acquired infection outbreaks. *PLoS computational biology*, 15(9):e1007284, 2019.

Albert, R. and Barabási, A.-L. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

Austin, J., Johnson, D. D., Ho, J., Tarlow, D., and van den Berg, R. Structured denoising diffusion models in discrete state-spaces. *Advances in Neural Information Processing Systems*, 34:17981–17993, 2021.

Cao, N. D. and Kipf, T. Molgan: An implicit generative model for small molecular graphs. *ICML 2018 workshop on Theoretical Foundations and Applications of Deep Generative Models*, (arXiv:1805.11973), 2022. doi: 10.48550/arXiv.1805.11973.

Chen, X., Han, X., Hu, J., Ruiz, F., and Liu, L. Order matters: Probabilistic modeling of node sequence for graph generation. In *International Conference on Machine Learning*, pp. 1630–1639. PMLR, 2021.

Costa, F. and De Grave, K. Fast neighborhood subgraph pairwise distance kernel. In *ICML*, 2010.

Dai, H., Nazi, A., Li, Y., Dai, B., and Schuurmans, D. Scalable deep generative modeling for sparse graphs. In *International conference on machine learning*, pp. 2302–2312. PMLR, 2020.

De Bortoli, V., Thornton, J., Heng, J., and Doucet, A. Diffusion schrödinger bridge with applications to score-based generative modeling. *Advances in Neural Information Processing Systems*, 34:17695–17709, 2021.

Du, Y., Wang, S., Guo, X., Cao, H., Hu, S., Jiang, J., Varala, A., Angirekula, A., and Zhao, L. Graphgt: Machine learning datasets for graph generation and transformation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

Erdos, P., Rényi, A., et al. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

Gonzalez-Malerva, L., Park, J., Zou, L., Hu, Y., Moradpour, Z., Pearlberg, J., Sawyer, J., Stevens, H., Harlow, E., and LaBaer, J. High-throughput ectopic expression screen for tamoxifen resistance identifies an atypical kinase that blocks autophagy. *Proceedings of the National Academy of Sciences*, 108(5):2058–2063, 2011.

Grover, A., Zweig, A., and Ermon, S. Graphite: Iterative generative modeling of graphs. In *International conference on machine learning*, pp. 2434–2444. PMLR, 2019.

Guo, X., Du, Y., and Zhao, L. Deep generative models for spatial networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 505–515, 2021.

Haefeli, K. K., Martinkus, K., Perraudin, N., and Wattenhofer, R. Diffusion models for graphs benefit from discrete state spaces. *arXiv preprint arXiv:2210.01549*, 2022.

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

Hoogeboom, E., Nielsen, D., Jaini, P., Forré, P., and Welling, M. Argmax flows and multinomial diffusion: Learning categorical distributions. *Advances in Neural Information Processing Systems*, 34:12454–12465, 2021.

Hoogeboom, E., Gritsenko, A. A., Bastings, J., Poole, B., van den Berg, R., and Salimans, T. Autoregressive diffusion models. In *International Conference on Learning Representations*, 2022.

Irwin, J. J., Sterling, T., Mysinger, M. M., Bolstad, E. S., and Coleman, R. G. Zinc: a free tool to discover chemistry for biology. *Journal of chemical information and modeling*, 52(7):1757–1768, 2012.

Jang, H., Justice, S., Polgreen, P. M., Segre, A. M., Sewell, D. K., and Pemmaraju, S. V. Evaluating architectural changes to alter pathogen dynamics in a dialysis unit. In *2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pp. 961–968. IEEE, 2019.

Jin, W., Barzilay, R., and Jaakkola, T. Junction tree variational autoencoder for molecular graph generation. In *Proceedings of the International Conference on Machine Learning*, pp. 2323–2332, 2018.

Jin, W., Barzilay, R., and Jaakkola, T. Hierarchical generation of molecular graphs using structural motifs. In *International conference on machine learning*, pp. 4839–4848. PMLR, 2020.

Jo, J., Lee, S., and Hwang, S. J. Score-based generative modeling of graphs via the system of stochastic differential equations. *arXiv preprint arXiv:2202.02514*, 2022.

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

Liao, R., Li, Y., Song, Y., Wang, S., Hamilton, W., Duvenaud, D. K., Urtasun, R., and Zemel, R. Efficient graph generation with graph recurrent attention networks. *Advances in neural information processing systems*, 32, 2019.

Liu, M., Yan, K., Oztekin, B., and Ji, S. Graphebm: Molecular graph generation with energy-based models. In *Energy Based Models Workshop-ICLR 2021*, 2021.

Liu, Q., Allamanis, M., Brockschmidt, M., and Gaunt, A. Constrained graph variational autoencoders for molecule design. *Advances in neural information processing systems*, 31, 2018.

Lucas, J., Tucker, G., Grosse, R., and Norouzi, M. Understanding posterior collapse in generative latent variable models. 2019a.

Lucas, J., Tucker, G., Grosse, R. B., and Norouzi, M. Don't blame the elbo! a linear vae perspective on posterior collapse. *Advances in Neural Information Processing Systems*, 32, 2019b.

Luo, Y., Yan, K., and Ji, S. Graphdf: A discrete flow model for molecular graph generation. In *International Conference on Machine Learning*, pp. 7192–7203. PMLR, 2021.

Ma, T., Chen, J., and Xiao, C. Constrained generation of semantically valid graphs via regularizing variational autoencoders. *Advances in Neural Information Processing Systems*, 31, 2018.

Madhawa, K., Ishiguro, K., Nakago, K., and Abe, M. Graphnvp: An invertible flow model for generating molecular graphs. *arXiv preprint arXiv:1905.11600*, 2019.

Martinkus, K., Loukas, A., Perraudin, N., and Wattenhofer, R. Spectre: Spectral conditioning helps to overcome the expressivity limits of one-shot graph generators. In *International Conference on Machine Learning*, 2022.

Maziarka, Ł., Pocha, A., Kaczmarczyk, J., Rataj, K., Danel, T., and Warchoł, M. Mol-cyclegan: a generative model for molecular optimization. *Journal of Cheminformatics*, 12(1):1–18, 2020.

Niu, C., Song, Y., Song, J., Zhao, S., Grover, A., and Ermon, S. Permutation invariant graph generation via score-based generative modeling. In *International Conference on Artificial Intelligence and Statistics*, pp. 4474–4484. PMLR, 2020.

Preuer, K., Renz, P., Unterthiner, T., Hochreiter, S., and Klambauer, G. Fréchet chemnet distance: a metric for generative models for molecules in drug discovery. *Journal of chemical information and modeling*, 58(9):1736–1741, 2018.

Ramakrishnan, R., Dral, P. O., Rupp, M., and Von Lilienfeld, O. A. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1(1):1–7, 2014.

Schomburg, I., Chang, A., Ebeling, C., Gremse, M., Heldt, C., Huhn, G., and Schomburg, D. Brenda, the enzyme database: updates and major new developments. *Nucleic acids research*, 32(suppl_1):D431–D433, 2004.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Shi, C., Xu, M., Zhu, Z., Zhang, W., Zhang, M., and Tang, J. Graphaf: a flow-based autoregressive model for molecular graph generation. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1esMkHYPr.

Simonovsky, M. and Komodakis, N. Graphvae: Towards generation of small graphs using variational autoencoders. In *International conference on artificial neural networks*, pp. 412–422. Springer, 2018.

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pp. 2256–2265. PMLR, 2015.

Song, Y. and Ermon, S. Generative modeling by estimating gradients of the data distribution. *Advances in Neural Information Processing Systems*, 32, 2019.

Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., and Poole, B. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Vignac, C., Krawczuk, I., Siraudin, A., Wang, B., Cevher, V., and Frossard, P. Digress: Discrete denoising diffusion for graph generation. *arXiv preprint arXiv:2209.14734*, 2022.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

You, J., Liu, B., Ying, Z., Pande, V., and Leskovec, J. Graph convolutional policy network for goal-directed molecular graph generation. *Advances in neural information processing systems*, 31, 2018a.

You, J., Ying, R., Ren, X., Hamilton, W., and Leskovec, J. Graphrnn: Generating realistic graphs with deep autoregressive models. In *International conference on machine learning*, pp. 5708–5717. PMLR, 2018b.

Yu, H., Sun, X., Solvang, W. D., and Zhao, X. Reverse logistics network design for effective management of medical waste in epidemic outbreaks: Insights from the coronavirus disease 2019 (covid-19) outbreak in wuhan (china). *International journal of environmental research and public health*, 17(5):1770, 2020.

Zang, C. and Wang, F. Moflow: an invertible flow model for generating molecular graphs. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 617–626, 2020.

Zhu, Y., Du, Y., Wang, Y., Xu, Y., Zhang, J., Liu, Q., and Wu, S. A survey on deep graph generation: Methods and applications. In *The First Learning on Graphs Conference*.

A. Appendix

A.1. Visualization of generation ordering

Figure 3 shows the graph generative process of GRAPHARM and OA-ARDM. As we can see, GRAPHARM first generates one community and then moves to another; while OA-ARDM randomly generates the graph and fails to capture the underlying graph distribution.

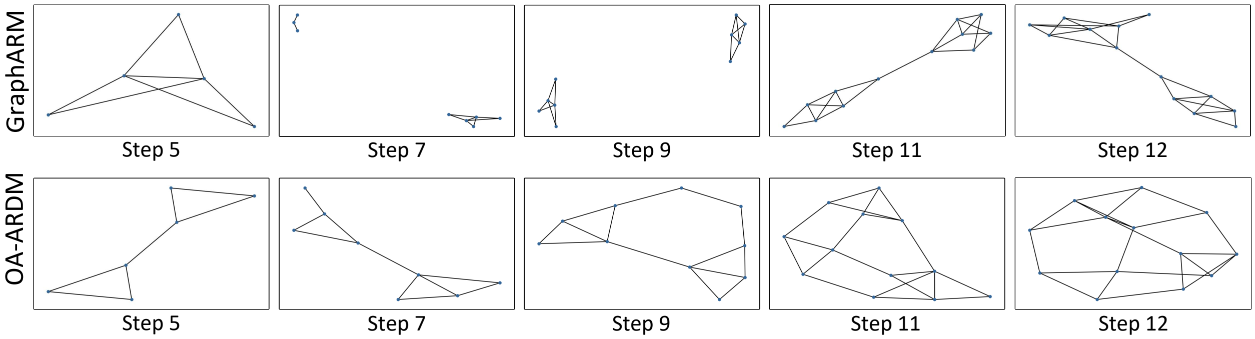

Figure 3. The graph generative process of GRAPHARM and OA-ARDM for community generation. As we can see, GRAPHARM first generates one community and then adds another one, which show that GRAPHARM captures graph structural topology for generation. In contrast, OA-ARDM generates the graph with a random order.

A.2. Evaluation of Negative Log likelihood

	Community-small	Breast
GRAN	23.04	247.18
OM	**16.82**	**187.22**
EDP-GNN	N/A	N/A
GDSS	N/A	N/A
GRAPHARM	**16.21**	**191.05**

Table 5. Test set negative log-likelihood (NLL) on Community-small and Breast datasets. N/A represents the model cannot provide the NLL.

We further evaluate the expected negative log-likelihood (NLL) across node permutations on the test sets. For GRAN, we sample 1000 node permutations from the uniform distribution. For OM and GRAPHARM, we sample 1000 node permutations from the ordering network. Table 5 shows the expected NLL on the test sets for community-small and Breast. As shown, GRAPHARM can achieve competitive results with OM and outperform GRAN by large margins. This is because both OM and GRAPHARM learn a data-dependent node ordering distribution; sampling from this distribution is more sample efficient than the uniform distribution. Though GDSS and EDP-GNN are also diffusion-based graph generative models, they cannot provide the likelihood. Note that GDSS involves a system of SDEs and we cannot directly use the equation from (Song et al., 2021) to compute the likelihood. To compute the likelihood, GDSS needs to derive the probability ODE flow that induces the same marginal probability of the system of SDEs. However, such a derivation is non-trivial and the GDSS paper did not provide it. This can be a drawback in some density-based downstream tasks, *e.g.*, outlier detection.

A.3. Connection Between Autoregressive Diffusion and Absorbing Diffusion

While ARDM appears different from classic diffusion, it amounts to absorbing diffusion with continuous time limit. Starting from state $\boldsymbol{x}_0$, we can define a continuous-time absorbing process, where each element $\boldsymbol{x}_t^{(i)}$ independently decays into an absorbing state with continuous-time probabilities $\alpha(t)$. This stochastic process is equivalent as using a finite set of D random transition times $\{\tau_i\}_{i=1}^{D}$ for recording the time where $\boldsymbol{x}_t^{(i)}$ was absorbed. It was shown by (Hoogeboom et al., 2022) that modeling the reverse generation of this process does not need to be conditioned on the precise values of the transition times τ_i. Hence, when training the reverse generative model, we only need to model $\boldsymbol{x}_{\tau_i}$ based on $\boldsymbol{x}_{\tau_{i+1}}$ while ignoring τ_i. This allows for writing the variational lower bound (VLB) of likelihood as an expectation over an uniform ordering in an autoregressive form:

$$\log p(\boldsymbol{x}_0) \geq \mathbb{E}_{\sigma \sim \mathcal{U}(S_d)} \sum_{i}^{D} \log p(\boldsymbol{x}_{\sigma(i)} | \boldsymbol{x}_{\sigma(>i)}). \tag{5}$$

A.4. Derivation of Eq. 3

The diffusion process assigns a unique ID to each node of G_0. Therefore, there is a one-to-one mapping between $G_{0:n}$ and $\sigma_{1:n}$, which leads to $p(\sigma_{1:n}|G_{0:n}) = 1$. Then, we have the following equations:

$$p_\theta(G_{1:n}, \sigma_{1:n}) = p_\theta(G_{1:n}), \tag{6}$$

$$q_\phi(G_{1:n}|G_0) = q_\phi(\sigma_{1:n}|G_0). \tag{7}$$

Then, our VLB can be written as:

$$\begin{aligned}
\log p_\theta(G_0) =& \log \left(\int p(G_{0:n}) \frac{q_\phi(G_{1:n}|G_0)}{q_\phi(G_{1:n}|G_0)} dG_{1:n} \right) \\
\geq& \mathbb{E}_{q_\phi(G_{1:n}|G_0)} \left(\log p(G_{1:n}) + \log p(G_0|G_{1:n}) - \log q_\phi(G_{1:n}|G_0) \right) \\
=& \mathbb{E}_{q_\phi(\sigma_{1:n}|G_0)} \left(\log p(G_{1:n}, \sigma_{1:n}) + \log p(G_0|G_{1:n}) - \log q_\phi(G_{1:n}|G_0) \right) \\
=& \mathbb{E}_{q_\phi(\sigma_{1:n}|G_0)} (\log p(G_{1:n-1}|\sigma_{1:n}) p(\sigma_{1:n}|G_n) p(G_n) + \log p(G_0|G_{1:n}) - \log q_\phi(\sigma_{1:n}|G_0)) \\
=& \mathbb{E}_{q_\phi(\sigma_{1:n}|G_0)} (\log p(G_{1:n-1}|\sigma_{1:n}) + \log p(G_0|G_{1:n}) + \log p(\sigma_{1:n}|G_n) + \cancelto{0}{\log p(G_n)} - \log q_\phi(\sigma_{1:n}|G_0)) \\
=& \mathbb{E}_{q_\phi(\sigma_{1:n}|G_0)} \left(\log p(G_{0:n-1}|\sigma_{1:n}, G_n) + \log p(\sigma_{1:n}|G_n) - \log q_\phi(\sigma_{1:n}|G_0) \right) \\
=& \mathbb{E}_{q_\phi(\sigma_{1:n}|G_0)} \sum_{t=0}^{n-1} \log p_\theta(G_t|G_{t+1}, \sigma_{t+1}) - \mathrm{KL}(q_\phi(\sigma_{1:n}|G_0) | p_{\theta'}(\sigma_{1:n}|G_n)) \\
=& \mathbb{E}_{q_\phi(\sigma_{1:n}|G_0)} \sum_{t=0}^{n-1} \log p_\theta(G_t|G_{t+1}) - \mathrm{KL}(q_\phi(\sigma_{1:n}|G_0) | p_{\theta'}(\sigma_{1:n}|G_n)).
\end{aligned} \tag{8}$$

We have $\log p(G_n) = 0$ because G_n is a deterministic graph wherein all the nodes are masked. In the last line, we omit the σ_t in $p_\theta(G_t|G_{t+1})$ because G_t together with G_{t+1} contains the information that we want to predict the node v_{σ_t} and its edges with previously denoised nodes.

A.5. Differences from OM

The previous works (Chen et al., 2021) first use the node ordering σ to permute the adjacency matrix and then autoregressively grow the permuted adjacency matrix A^σ row by row. However, the permuted adjacency matrix A^σ does not contain the exact node ordering information σ since multiple node orders can lead to the same adjacency matrix. In OM, $p_\theta(A^\sigma_{1:n}, \sigma_{1:n})$ is factorized as:

$$p_\theta\left(A^\sigma_{1:n}, \sigma_{1:n}\right) = p_\theta\left(A^\sigma_{1:n}\right) p_\theta\left(\sigma_{1:n} \mid A^\sigma_{1:n}\right).$$

As there are multiple $\sigma_{1:n}$ that lead to $A^s_{1:n}$, OM inevitably needs to compute the graph automorphism in their VLB.

Why do we not need to compute the graph automorphism as (Chen et al., 2021)? This is because, in our framework, the node ordering $\sigma_{1:n}$ is sampled in the forward diffusion process which essentially assigns a unique ID to each node of G_0. This creates a one-to-one mapping between $G_{0:n}$ and $\sigma_{1:n}$. In the generation procedure, our starting point is the masked graph G_n with n nodes; our generator network sequentially denoises the node v_{σ_t} in G_n in the reverse order of the diffusion ordering during training.

Different from OM, we factorize $p_\theta\left(G_{1:n}\right)$ (line 4 of Eq. 8) as:

$$p\left(G_{1:n}, \sigma_{1:n}\right) = p\left(G_{1:n-1} \mid \sigma_{1:n}\right) p\left(\sigma_{1:n} \mid G_n\right) p\left(G_n\right).$$

This factorization leads to the following consequences / benefits:

- The challenge of modeling the node ordering variable is now pushed into the term $p\left(\sigma_{1:n} \mid G_n\right)$. This is another key reason that we do not have the graph automorphism term in our VLB, instead we have a KL term. Now how do we learn this conditional distribution? It turns out we do not need to! While the generation ordering network is required for non-graph data such as text to determine which token to unmask at test time, it is not needed for graph generation due to node permutation invariance. The first term in Eq. 2 will encourage the denoising network $p_\theta(G_t|G_{t+1})$ to predict the node and edge types in the exact reverse ordering of the diffusion process, thus the denoising network itself can be a proxy of the generation ordering. Due to permutation invariance, we can simply replace any masked node and its masked edges with the predicted node and edge types at each time step.

- Still because of the factorization, our generator network $p_\theta(G_{0:n-1}|G_n, \sigma_{1:n})$ models the graph generation *conditioned on* a fixed node ordering. In other words, given an oracle ordering of how the nodes should be generated (which is given by the diffusion ordering network q), we use the generator network to model how the graph sequences are generated.

A.6. Training algorithm of GRAPHARM

We use a reinforcement learning (RL) procedure by sampling multiple diffusion trajectories, thereby enabling training both the diffusion ordering network $q_\phi(\sigma|G_0)$ and the denoising network $p_\theta(G_t|G_{t+1})$ using gradient descent.

Specifically, at each training iteration, we explore the diffusion ordering network by creating M diffusion trajectories for each training graph $G_0^{(i)}$. Each trajectory is a sequence of graphs $\{G_t^{i,m}\}_{1\le t\le n}$ where the node decay ordering $\sigma^{i,m}$ is sampled from $q_\phi(\sigma|G_0^{i,m})$. For each trajectory, we sample T time steps. The denoising network $p_\theta(G_t|G_{t+1})$ is then trained to minimize the negative VLB using stochastic gradient descent (SGD):

$$\triangle\theta \leftarrow \frac{\eta_1}{M}\nabla \sum_{i\in\mathcal{B}_{\text{train}}}\sum_{m,t}\sum_{k\in\sigma_{(\le t)}} \frac{n_i w_k^{i,m}}{T}\log p_\theta(O_{v_k}^{\sigma(>t)}|G_{t+1}^{i,m}),$$

where $\mathcal{B}_{\text{train}}$ is the a minibatch sampled from the training data and $w_k^{i,m} = q_\phi(\sigma_t^{i,m} = k|G_0^i, \sigma_{(<t)}^{i,m})$.

To evaluate the current diffusion ordering network, we create M trajectories for each validation graph and compute the negative VLB of the denoising network to obtain the corresponding rewards $R^{i,m} = -\sum_t \sum_{k\in\sigma_{(\le t)}} \frac{n_i}{T} w_k^{i,m} \text{log} p_\theta(O_{v_k}^{\sigma(>t)}|G_{t+1}^{i,m})$. Then, the diffusion ordering network can be updated with common RL optimization methods, *e.g.*, the REINFORCE algorithm (Williams, 1992):

$$\triangle\phi \leftarrow \frac{\eta_2}{M}\sum_{i\in\mathcal{B}_{\text{val}}}\sum_{m} R^{i,m}\nabla\log q_\phi(\sigma|G_0^{i,m}). \tag{9}$$

The detailed training procedure is summarized in Algorithm 1.

A.7. Implementation Details on Generic Graphs

Model optimization: We use ADAM with $\beta_1 = 0.9$ and $\beta = 0.999$ as the optimizer. The learning rate is set for 10^{-4} and 5×10^{-4} for the denoising network and diffusion ordering network respectively on all the datasets. We perform model selection based on the average MMD of the three metrics on the validation set.

Network architecture: For fair comparison, our denoising network uses the same graph attention network architecture as the baseline GRAN which has 7 layers and hidden dimensions 128; the diffusion ordering network uses the same graph attention network architecture as the baseline OA which is a vallina GAT (Veličković et al., 2018) that has 3 layers with 6 attention heads and residual connections. The hidden dimension is set to 32. As OM is compatible with any existing autoregressive methods, we use the strongest autoregressive baseline GRAN as its backbone.

Hyper-parameters: We set the number of trajectories M as 4 for all the datasets. Both GRAPHARM and GRAN use 20 as the number of Bernoulli mixtures. For GRAN, we use block size 1 and stride 1 to achieve the best generation performance. For GDSS, we choose the best signal-to-noise ratio (SNR) from $\{0.05, 0.1, 0.15, 0.2\}$ and scale coefficient from $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$ based on the average MMD of degree, clustering coefficient and orbit as in (Jo et al., 2022). For EDP-GNN, we use 6 noise levels $\{\sigma_i\}_{i=1}^{L} = [1.6, 0.8, 0.6, 0.4, 0.2, 0.1]$ as suggested in the original work (Niu et al., 2020). For OM, we use 4 as the sample size for variational inference as suggested in the original work (Chen et al., 2021)

Algorithm 1 Training procedure of GraphArm

```
Require: Diffusion ordering network $q_\phi(\sigma|G_0)$, Denoising network $p_\theta(G_t|G_{t+1})$
for # training iterations do
    Sample a minibatch $\mathcal{B}$ from the training set
    for each $i \in \mathcal{B}_{\text{train}}$ do
        for each $m \in [1, M]$ do
            $\sigma_1^{i,m}, \cdots, \sigma_{n_i}^{i,m} \sim q_\phi(\sigma|G_0)$
            Obtain corresponding diffusion trajectories $\{G_t^{i,m}\}_{1\leq t\leq n}$
        end for
        Sample $T$ time steps from $\mathcal{U}_n$
        $\theta_j \leftarrow \theta_{j-1} - \frac{\eta_1}{M}\nabla\sum_{i\in\mathcal{B}_{\text{train}}}\sum_{m,t}\sum_{k\in\sigma_{(\leq t)}}\frac{n_i w_k^{i,m}}{T}\log p_\theta(O_{v_k}^{\sigma(>t)}|G_{t+1}^{i,m})$
    end for
    Sample a minibatch from the validation set
    for each $i \in \mathcal{B}_{\text{val}}$ do
        for each $m \in [1, M]$ do
            $\sigma_1^{i,m}, \cdots, \sigma_{n_i}^{i,m} \sim q_\phi(\sigma|G_0)$
            Obtain corresponding diffusion trajectories $\{G_t^{i,m}\}_{1\leq t\leq n}$
        end for
        Sample $T$ time steps from $\mathcal{U}_n$
        Compute the reward $R^{i,m} = -\sum_t\sum_{k\in\sigma_{(\leq t)}}\frac{n_i}{T}w_k^{i,m}\text{log}p_\theta(O_{v_k}^{\sigma(>t)}|G_{t+1}^{i,m})$
        $\triangle\phi \leftarrow \frac{\eta_2}{M}\sum_{i\in\mathcal{B}_{\text{val}}}\sum_m R^{i,m}\nabla\log q_\phi(\sigma|G_0^{i,m})$.
    end for
end for
```

A.8. Baselines and Implementation Details on Molecule Datasets

Baselines: We compare GraphArm with the following baselines: GraphAF (Shi et al., 2020) and GraphDF (Luo et al., 2021) are flow-based autoregressive models. GraphEBM (Liu et al., 2021) is a one-shot energy-based model. MoFlow (Zang & Wang, 2020) is a one-shot normalizing flow model. SPECTRE (Martinkus et al., 2022) is a one-shot model based on GAN from a spectral perspective. DiGress (Vignac et al., 2022), EDP-GNN (Niu et al., 2020) and GDSS (Jo et al., 2022) are scored-based one-shot methods.

Network architecture:

Denosing network:

We first use a one linear layer MLP to project the node type and edge type into the continuous embedding space.

$$\boldsymbol{h}_{v_i}^0 = \text{MLP}(v_i), \quad \boldsymbol{h}_{e_{v_i,v_j}} = \text{MLP}(e_{v_i,v_j})$$

Then, the l-th round of message passing is implemented as:

$$\boldsymbol{m}_{i,j}^l = f([\boldsymbol{h}_{v_i}^l|\boldsymbol{h}_{v_j}^l|\boldsymbol{h}_{e_{v_i,v_j}}]), \quad a_{i,j}^l = \text{Sigmoid}(g([\boldsymbol{h}_{v_i}^l|\boldsymbol{h}_{v_j}^l|\boldsymbol{h}_{e_{v_i,v_j}}])),$$

$$\boldsymbol{h}_{v_i}^{l+1} = \text{GRU}(\boldsymbol{h}_{v_i}^l, \sum_{j\in\mathcal{N}(i)} a_{i,j}^l \boldsymbol{m}_{i,j}^l),$$

where $[\cdot|\cdot]$ is the concatenate operation. Both f and g are 2-layer MLPs with ReLU nonlinearities and hidden dimension 256.

After $L = 5$ rounds of message passing, we obtain the final node representation vectors $\boldsymbol{h}_{v_i}^L$ for each node. With the average pooling operation, we can further obtain the graph level representation $\boldsymbol{h}_G^L$.

Then the prediction of node v_{σ_t} follows a multinomial distribution:

$$p(v_{\sigma_t}|G_{t+1}) = \text{Softmax}(\text{MLP}_n([\boldsymbol{h}_G^L|\boldsymbol{h}_{v_{\sigma_t}}]),$$

where we implement MLP_n as a 2-layer MLPs with ReLU nonlinearities and hidden dimension 256.

The prediction of edges between v_{σ_t} and all the previously denoised nodes $\{v_j\}_{j\in\sigma_{(>t)}}$ follows a mixture of Multinomial distribution:

$$p(e_{v_t,v_j}|G_{t+1}) = \sum_{k=1}^{K} \alpha_k \text{Softmax}(\text{MLP}_{e_k}([\boldsymbol{h}_G^L|\boldsymbol{h}_{v_t}|\boldsymbol{h}_{v_j}]),$$

$$\alpha_1, \cdots, \alpha_K = \text{Softmax}(\sum_{j\in\sigma_{(>t)}} \text{MLP}_{\alpha}([\boldsymbol{h}_G^L|\boldsymbol{h}_{v_i}|\boldsymbol{h}_{v_j}])).$$

Both MLP_α and MLP_{e_k} are 2-layer MLPs with ReLU nonlinearities and hidden dimension 256. We use $K = 20$ in both experiments.

Diffusion ordering network: We use a 3-layer relational graph convolutional network with hidden dimension 256.

Model optimization: We use ADAM with $\beta_1 = 0.9$ and $\beta = 0.999$ as the optimizer. The learning rate is set for 10^{-3} and 5×10^{-2} for the denoising network and diffusion ordering network respectively on both the datasets.

A.9. Experiment Setup for Constrained Graph Generation

We use the Caveman dataset for the constrained generation experiment. We set the constraint as that the maximum node degree is no larger than 6. To generate graphs under this constraint with GRAPHARM, we add a degree checking into the generation process. Specifically, when generating a new node and its connecting edges, we first check whether the constraint will be violated on previous nodes as new edges are added to them. If so, the corresponding edges will be dropped; otherwise we proceed to check the constraint on the new generated node and randomly remove the extra edges that exceed the limit.

A.10. Dataset Statistics

We evaluate the performance of GRAPHARM on six diverse graph generation benchmarks, covering both synthetic and real-world graphs from different domains: (1) Community-small (You et al., 2018b), containing 100 graphs randomly generated community graphs with $12 \leq |V| \leq 20$; (2) Caveman (You, 2018), containing 200 caveman graphs synthetically generated with $5 \leq |V| \leq 10$; (3) Cora, containing 200 sub-graphs with $9 \leq |V| \leq 87$, extracted from Cora network (Sen et al., 2008) using random walk; (4) Breast, including 100 chemical graphs with $12 \leq |V| \leq 18$, sampled from (Gonzalez-Malerva et al., 2011); (5) Enzymes, including 563 protein graphs with $10 \leq |V| \leq 125$ from BRENDA database (Schomburg et al., 2004); (6) Ego-small, containing 200 small sub-graphs with $4 \leq |V| \leq 18$ sampled from Citeseer Network Dataset (Sen et al., 2008). For each dataset, we use 80% of the graphs as training set and the rest 20% as test sets. Following (Liao et al., 2019), we randomly select 25% from the training data as the validation set.

Table 6 provides the statistics of QM9 and ZINC250k datasets used in the molecule generation tasks.

Dataset	Number of graphs	Number of nodes	Number of node types	Number of edge types
QM9	133,885	$1 \leq \lvert V\rvert \leq 9$	4	3
ZINC250k	249,455	$6 \leq \lvert V\rvert \leq 38$	9	3

Table 6. Statistics of QM9 and ZINC250k datasets used in the molecule generation tasks.

A.11. Verification of the Denoising ordering

In Eq. 3, we do not learn a separate generation ordering but directly replace the masked node and edges with the predictions of the denoising network at test time. This is because graph is node permutation invariant and the first term in Eq. 3 encourages the denoising model to predict the node and edge types in the reverse of the diffusion ordering. To evaluate the generation ordering directly obtained from the denoising network, we compute the KL-divergence between the diffusion ordering and the denoising ordering at each time step and then sum together, i.e., Approximation error= $\sum_t \text{KL}\left(q(\sigma_t|G_0, \sigma_{(<t)})|p(\sigma_t|G_t)\right)$. Specifically, at each time step t, we use GRAPHARM to first generate a graph G_0 and record the corresponding node generation ordering σ. Since we do not explicitly model distribution of the node index during the generation procedure, we determine the index of the generated node in G_0 by its connectivity to the unmasked nodes in G_t. Finally, we forward the generation network multiple times to obtain an empirical distribution for $p(\sigma_t|G_t)$. Table 7 provides the approximation error versus the training iterations. As we can see, with the training progresses, the approximation error indeed becomes smaller and approaches to zero.

Training iterations	0	100	200	500	2000	3000
Appriximation error	0.343	0.262	0.131	0.093	0.061	0.049

Table 7. Approximation error between the generation ordering and the diffusion ordering on the community-small dataset.

A.12. Visualization of Generated Samples

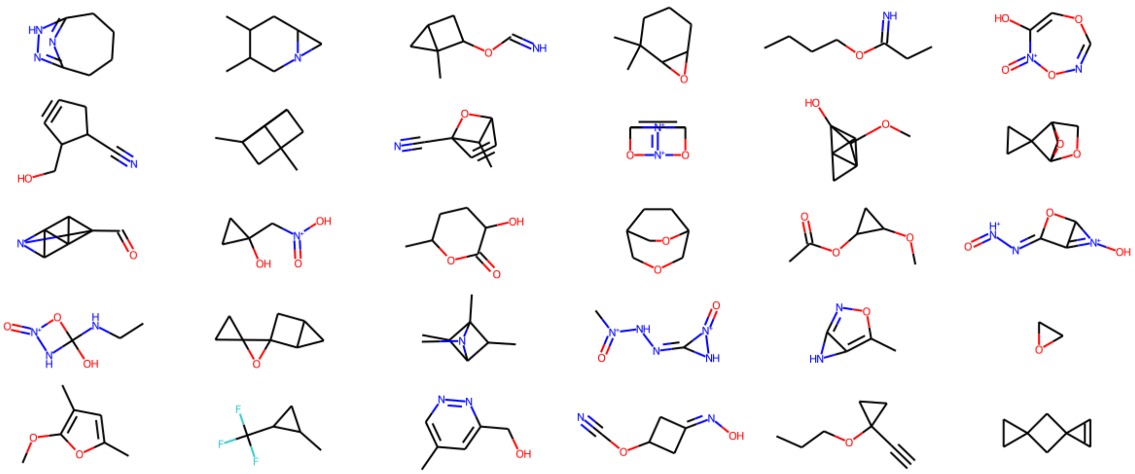

Figure 4. Visualization of generated samples of GRAPHARM on QM9 dataset

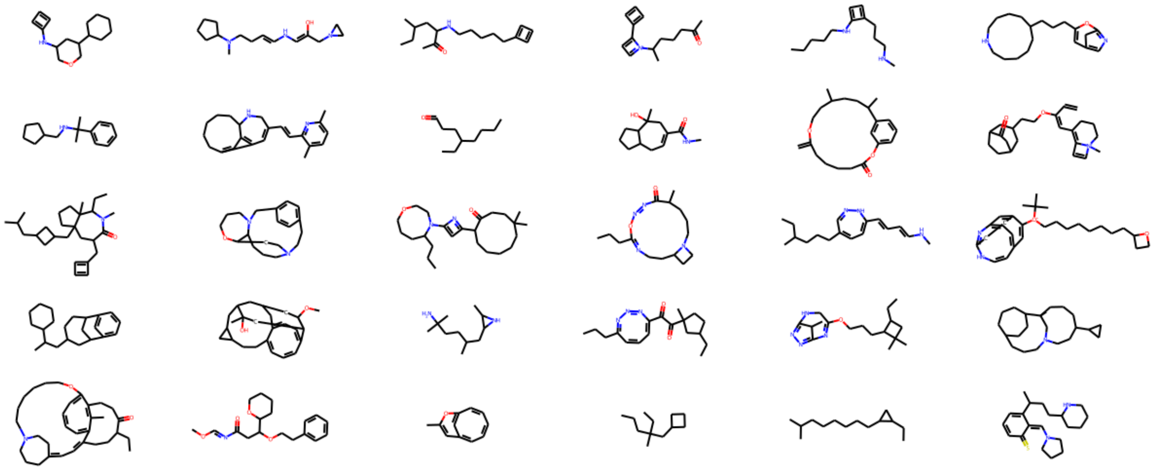

Figure 5. Visualization of generated samples of GRAPHARM on ZINC250K dataset