Efficient and Degree-Guided Graph Generation via Discrete Diffusion Modeling

Xiaohui Chen [1] **Jiaxing He** [1] **Xu Han** [1] **Li-Ping Liu** [1]

Abstract

Diffusion-based graph generative models are effective in generating high-quality small graphs. However, it is hard to scale them to large graphs that contain thousands of nodes. In this work, we propose EDGE, a new diffusion-based graph generative model that addresses generative tasks for large graphs. The model is developed by reversing a discrete diffusion process that randomly removes edges until obtaining an empty graph. It leverages graph sparsity in the diffusion process to improve computational efficiency. In particular, EDGE only focuses on a small portion of graph nodes and only adds edges between these nodes. Without compromising modeling ability, it makes much fewer edge predictions than previous diffusion-based generative models. Furthermore, EDGE can explicitly model the node degrees of training graphs and then gain performance improvement in capturing graph statistics. The empirical study shows that EDGE is much more efficient than competing methods and can generate large graphs with thousands of nodes. It also outperforms baseline models in generation quality: graphs generated by the proposed model have graph statistics more similar to those of training graphs.

1. Introduction

There is a long history of using random graph models (Newman et al., 2002) to model large graphs. Traditional models such as Erdős-Rényi (ER) model (Erdos et al., 1960), Stochastic-Block Model (SBM) (Holland et al., 1983), and Exponential-family Random Graph Models (Lusher et al., 2013) are often used to model existing graph data and focus on prescribed graph structures. Besides modeling existing data, one interesting problem is to generate new graphs to simulate existing ones (Ying & Wu, 2009), which has applications such as network data sharing. In generative tasks (Chakrabarti & Faloutsos, 2006), traditional models often fall short in describing complex structures. A promising direction is to use deep neural models to generate large graphs.

[1]Department of Computer Science, Tufts University, Medford, MA, USA. Correspondence to: Xiaohui Chen <xiaohui.chen@tufts.edu>, Li-Ping Liu <liping.liu@tufts.edu>.

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

There are only a few deep generative models designed for generating large graphs: NetGAN (Bojchevski et al., 2018) and CELL (Rendsburg et al., 2020) are two examples. However, recent research (Chanpuriya et al., 2021) shows that these two models are edge-independent models and have a theoretical limitation: they cannot reproduce several important statistics (e.g. triangle counts and clustering coefficient) in their generated graphs unless they memorize the training graph. A list of other models (Chanpuriya et al., 2021) including Variational Graph Autoencoders (VGAE) (Kipf & Welling, 2016b) and GraphVAE (Simonovsky & Komodakis, 2018) are also edge-independent models and share the same limitation.

Diffusion-based generative models (Liu et al., 2019; Niu et al., 2020; Jo et al., 2022; Chen et al., 2022b) have gained success in modeling small graphs. These models generate a graph in multiple steps and are NOT edge-independent because edges generated in later steps depend on previously generated edges. They are more flexible than one-shot models (Kipf & Welling, 2016b; Madhawa et al., 2019; Lippe & Gavves, 2020), which directly predict an adjacency matrix in one step. They also have an advantage over autoregressive graph models (You et al., 2018; Liao et al., 2019), as diffusion-based models are invariant to node permutations and do not have long-term memory issues. However, diffusion-based models are only designed for tasks with small graphs (usually with less than one hundred nodes).

This work aims to scale diffusion-based generative models to large graphs. The major issue of a diffusion-based model is that it must compute a latent vector or a probability for each node pair in a graph at each diffusion step (Niu et al., 2020; Jo et al., 2022) – the computation cost is $O(TN^2)$ if the model generates a graph with N nodes using T steps. The learning task becomes challenging when N is large. At the same time, large graphs increase the difficulties for a model to capture global graph statistics such as clustering coefficients. As a result, the model performance degrades when the training graphs' sizes scale up.

We propose *Efficient and Degree-guided graph GEnerative model* (EDGE) based on a discrete diffusion process. The development of EDGE has three innovations. First, we encourage the sparsity of graphs in the diffusion process by setting the empty graph as the convergent "distribution".

Then the diffusion process only removes edges and can be viewed as an edge-removal process. The increased sparsity in graphs in the process dramatically reduces the computation – this is because the message-passing neural network (MPNN) (Kipf & Welling, 2016a) used in the generative model needs to run on these graphs, and their runtime is linear in the number of edges. Second, the generative model, which is the reverse of the edge-removal process, only predicts edges for a small portion of "active nodes" that have edge changes in the original edge-removal process. This strategy decreases the number of predictions of MPNN and also its computation time. More importantly, this new design is naturally derived from the aforementioned edge-removal process without modifying its forward transition probabilities. Third, we model the node degrees of training graphs explicitly. By characterizing the node degrees, the statistics of the generated graphs are much closer to training graphs. While other diffusion-based graph models struggle to even train or sample on large graphs, our approach can efficiently generate large graphs with desired statistical properties. We summarize our contributions as follows:

- we use empty graphs as the convergent distribution in a discrete diffusion process to reduce computation;
- we propose a new generative process that only predicts edges between a fraction of nodes in graphs;
- we explicitly model node degrees in the probabilistic framework to improve graph statistics of generated graphs; and
- we conduct an extensive empirical study and show that our method can efficiently generate large graphs with desired statistics.

2. Background

This work considers graph generative models that sample adjacency matrices to generate graphs. Let $\mathcal{A}^N$ denote the space of adjacency matrices of size N. We consider simple graphs without self-loops or multi-edges, so an adjacency matrix $\mathbf{A} \in \mathcal{A}^N$ is a binary symmetric matrix with a zero diagonal. A generative model defines a distribution over $\mathcal{A}^N$.

In this work, we construct a generative model based on a discrete diffusion process (Austin et al., 2021; Hoogeboom et al., 2021; Vignac et al., 2022). Let $\mathbf{A}^0$ denote a graph from the data, then the diffusion process defined by $q(\mathbf{A}^t|\mathbf{A}^{t-1})$ corrupts $\mathbf{A}^0$ in T steps and forms a trajectory $(\mathbf{A}^0, \mathbf{A}^1, \ldots, \mathbf{A}^T)$. We treat $(\mathbf{A}^1, \ldots, \mathbf{A}^T)$ as latent variables, then $q(\mathbf{A}^1, \ldots, \mathbf{A}^T|\mathbf{A}^0) = \prod_{t=1}^T q(\mathbf{A}^t|\mathbf{A}^{t-1})$. As $T \to \infty$, $q(\mathbf{A}^T)$ approaches a convergent distribution, which is often a simple one with easy samples. We often choose a large enough T so that $q(\mathbf{A}^T)$ is a good approximation of the convergent distribution.

We model these trajectories with a denoising model $p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t)$ parameterized by θ, then the model has a joint $p_\theta(\mathbf{A}^{0:T}) = p(\mathbf{A}^T)\prod_{t=1}^T p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t)$ and a marginal $p_\theta(\mathbf{A}^0)$ that describes the data distribution. Here $p(\mathbf{A}^T)$ is the convergent distribution in q.

Usually $q(\mathbf{A}^t|\mathbf{A}^{t-1})$ needs easy probability calculations. One choice is to treat each edge independently, and

$$q(\mathbf{A}^t|\mathbf{A}^{t-1}) = \prod_{i,j:i<j} \mathcal{B}(\mathbf{A}^t_{i,j}; (1-\beta_t)\mathbf{A}^{t-1}_{i,j} + \beta_t p) \quad (1)$$
$$:= \mathcal{B}(\mathbf{A}^t; (1-\beta_t)\mathbf{A}^{t-1} + \beta_t p).$$

Here $\mathcal{B}(x;\mu)$ represents the Bernoulli distribution over x with probability μ. We also use $\mathcal{B}(\mathbf{A};\boldsymbol{\mu})$ to represent the probability of independent Bernoulli variables arranged in a matrix. The diffusion rate β_t determines the probability of resampling the entry $\mathbf{A}^t_{i,j}$ from a Bernoulli distribution with probability p, instead of keeping the entry $\mathbf{A}^{t-1}_{i,j}$.

This diffusion process requires two special properties for model fitting. First, we can sample $\mathbf{A}^t$ at any time step t directly from $\mathbf{A}^0$. Let $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{\tau=1}^t \alpha_\tau$,

$$q(\mathbf{A}^t|\mathbf{A}^0) = \mathcal{B}(\mathbf{A}^t; \bar{\alpha}_t\mathbf{A}^0 + (1-\bar{\alpha}_t)p). \quad (2)$$

The diffusion rates β_t-s are defined in a way such that $\bar{\alpha}_T$ is almost 0, then $\mathbf{A}^T$ is almost independent from $\mathbf{A}^0$, i.e., $q(\mathbf{A}^T|\mathbf{A}^0) \approx p(\mathbf{A}^T) \equiv \mathcal{B}(\mathbf{A}^T; p)$. The configuration of β_t-s is called *noise scheduling*. In the context of graph generation, $p(\mathbf{A}^T)$ is the Erdős-Rényi graph model $G(N,p)$ (Erdos et al., 1960), with p being the probability of forming an edge between two nodes.

Second, we can compute the posterior of the forward transition when conditioning on $\mathbf{A}^0$:

$$q(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{A}^0) = \frac{q(\mathbf{A}^t|\mathbf{A}^{t-1})q(\mathbf{A}^{t-1}|\mathbf{A}^0)}{q(\mathbf{A}^t|\mathbf{A}^0)}. \quad (3)$$

Since all the terms on the right-hand side are known, the posterior can be computed analytically.

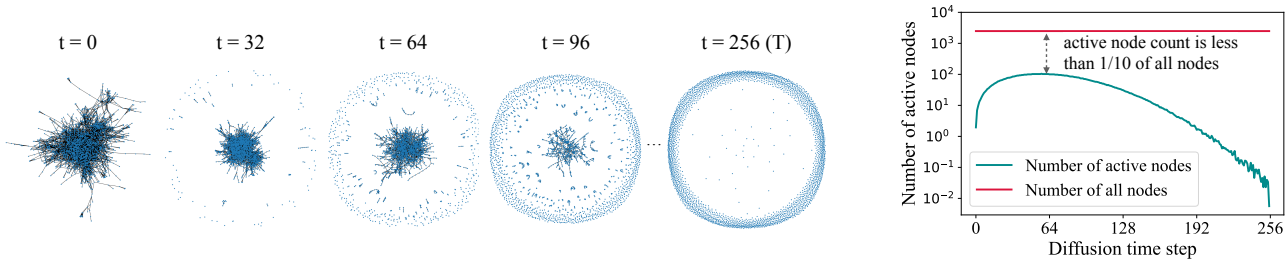
The generative model $p_\theta(\mathbf{A}^{0:T})$ is trained by maximizing a variational lower bound of $\log p_\theta(\mathbf{A}^0)$ (Ho et al., 2020; Hoogeboom et al., 2021; Austin et al., 2021). In an intuitive understanding, $p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t)$ is learned to match the posterior of the forward transition$q(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{A}^0)$.

During generation, we sample $\mathbf{A}^T \sim p(\mathbf{A}^T)$ and then "denoise" it iteratively with $p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t)$ to get an $\mathbf{A}^0$ sample.

3. Method

3.1. Diffuse graphs to empty graphs – a motivation

With the main purpose of computation efficiency, we advocate setting $p = 0$ and using $G(N, 0)$ as the convergent distribution. This configuration improves the sparsity of the adjacency matrices in diffusion trajectories, thus reducing



(a) An edge-removal process defined by $p = 0$

(b) Active nodes in the edge-removal process

Figure 1. Dynamics of a discrete diffusion process with $p = 0$ and "active" nodes in the process on the Cora dataset: (a) the diffusion process with $p = 0$ is an edge-removal process. The reverse of it is a generative procedure that constructs a graph by gradually adding edges to an empty graph. (b) under linear noise scheduling, the number of "active" nodes (that have their edges removed at a time step) is less than one-tenth of the total number of nodes.

computation. We consider the amount of computation in the denoising model $p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t)$ from two aspects: the computation on the input $\mathbf{A}^t$ and the number of entries to be predicted in the output $\mathbf{A}^{t-1}$.

We first consider the computation on the input side. We assume that the denoising model $p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t)$ is constructed with an MPNN. Suppose the input graph $\mathbf{A}^t$ has M^t edges, then a typical MPNN needs to perform $O(M^t)$ message-passing operations to compute node vectors – here we treat hidden sizes and the number of network layers as constants. The total number of message-passing operations over the trajectory is $O(\sum_{t=1}^T M^t)$. After some calculations, we show that

$$\sum_{t=1}^T M^t = M^0 \sum_{t=1}^T \bar{\alpha}_t + \frac{N(N-1)p}{2} \sum_{t=1}^T 1 - \bar{\alpha}_t. \quad (4)$$

By setting $p = 0$, we eliminate the second term and reduce the number of edges in graphs in the diffusion trajectory by a significant factor, then the MPNN will have much fewer message-passing operations.

We then analyze the number of entries we need to predict in the output $\mathbf{A}^{t-1}$. When $p = 0$, the forward process is an edge-removal process, and the degree of a node is non-increasing for any forward transition. A node with a degree change from $t-1$ to t is considered "active". When a node is inactive at $t-1$, all edges incident to this node is kept at t. Figure 1 shows the average number of active nodes for each forward transition. We observe that active nodes only take a small fraction of the total when the convergent distribution is $G(N, 0)$.

While a previous diffusion-based model makes predictions for all node pairs, the observation above indicates that we can save computation by making predictions only for pairs of active nodes. In particular, the denoising model can first infer which nodes are active in each step and then only predict edges between active nodes. Below we will develop such a model and only consider the diffusion process with $p = 0$.

3.2. A diffusion-based model that explicitly models active nodes

We treat the "active nodes" as latent variables $\mathbf{s}^{1:T}$ and incorporate them into both the forward and reverse processes. Let $\mathbf{d}^t = \deg(\mathbf{A}^t)$ be the node degree vector of $\mathbf{A}^t$, then $\mathbf{s}^t := \mathbb{1}[\mathbf{d}^{t-1} \neq \mathbf{d}^t]$ is a binary vector indicating whether nodes are active (having degree change from $t-1$ to t) or not from $t-1$ to t. In the following, we redefine the forward and reverse processes.

Forward process. With latent variables $\mathbf{s}^{1:T}$, we show that the forward process can be rewritten into the following decomposition:

$$q(\mathbf{A}^{1:T}, \mathbf{s}^{1:T}|\mathbf{A}^0) = \prod_{t=1}^T q(\mathbf{A}^t|\mathbf{A}^{t-1}) q(\mathbf{s}^t|\mathbf{A}^{t-1}, \mathbf{A}^t). \quad (5)$$

The forward process does not change by including $\mathbf{s}^{1:T}$ since the value of $\mathbf{s}^t$ is determined by $\mathbf{A}^{t-1}$ and $\mathbf{A}^t$. This allows us to use still the forward transition $q(\mathbf{A}^t|\mathbf{A}^{t-1})$ to draw the entire sequence.

Reverse process. We decompose the denoising model as follows:

$$p_\theta(\mathbf{A}^{0:T}, \mathbf{s}^{1:T}) = p(\mathbf{A}^T) \prod_{t=1}^T p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{s}^t) p_\theta(\mathbf{s}^t|\mathbf{A}^t). \quad (6)$$

Here both $p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{s}^t)$ and $p_\theta(\mathbf{s}^t|\mathbf{A}^t)$ are learnable distributions. Intuitively, the denoising model first predicts which nodes are active ($\mathbf{s}^t$) and then generates edges between them to obtain $\mathbf{A}^{t-1}$. Since we only predict edges between active nodes indicated by $\mathbf{s}^t$, all edges that incident inactive nodes are carried from $\mathbf{A}^t$ to $\mathbf{A}^{t-1}$ directly.

Our EDGE model is specified by (6). The generative framework is demonstrated in Figure 2.

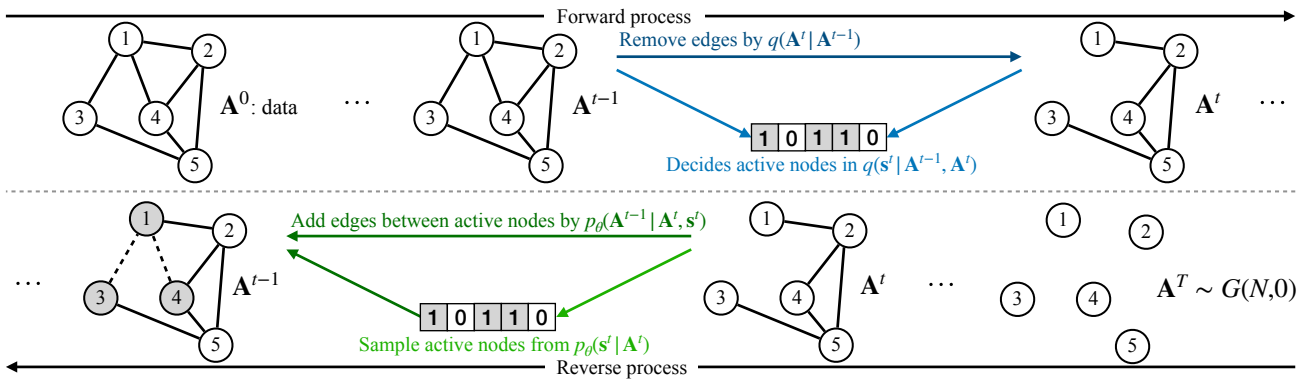


Figure 2. Forward and reverse processes. For the forward process, $\mathbf{A}^t$ is sampled from $q(\mathbf{A}^t|\mathbf{A}^{t-1})$, then $\mathbf{s}^t$ is deterministically generated given $\mathbf{A}^{t-1}$ and $\mathbf{A}^t$. For the reverse process, $\mathbf{s}^t$ is first sampled from a node selection distribution $p_\theta(\mathbf{s}^t|\mathbf{A}^t)$, then $\mathbf{A}^{t-1}$ is sampled from the parameterized distribution $p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{s}^t)$.

3.3. Learning the reverse process

We optimize the model parameters θ by maximizing the variational lower bound (VLB) of $\log p(\mathbf{A}^0)$. Following Sohl-Dickstein et al. (2015); Ho et al. (2020), the VLB is:

$$\mathcal{L}(\mathbf{A}^0;\theta) = \mathbb{E}_q\Big[\log \frac{p_\theta(\mathbf{A}^{0:T}, \mathbf{s}^{1:T})}{q(\mathbf{A}^{1:T}, \mathbf{s}^{1:T}|\mathbf{A}^0)}\Big] \tag{7}$$

$$= \mathbb{E}_q\Bigg[\log \frac{p(\mathbf{A}^T)}{q(\mathbf{A}^T|\mathbf{A}^0)} + \underbrace{\log p_\theta(\mathbf{A}^0|\mathbf{A}^1, \mathbf{s}^1)}_{\text{reconstruction term } \mathcal{L}_{\text{rec}}} +$$

$$\sum_{t=2}^{T} \underbrace{\log \frac{p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{s}^t)}{q(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{s}^t, \mathbf{A}^0)}}_{\text{edge prediction term } \mathcal{L}_{\text{edge}}(t)} + \sum_{t=1}^{T} \underbrace{\log \frac{p_\theta(\mathbf{s}^t|\mathbf{A}^t)}{q(\mathbf{s}^t|\mathbf{A}^t, \mathbf{A}^0)}}_{\text{node selection term } \mathcal{L}_{\text{node}}(t)}\Bigg].$$

Appendix B.1 shows detailed derivation. The first term contains no learnable parameters. The second term measures the reconstruction likelihood. For the edge prediction term $\mathcal{L}_{\text{edge}}(t)$, unlike Sohl-Dickstein et al. (2015); Ho et al. (2020), the posterior $q(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{s}^t, \mathbf{A}^0)$ is hard to compute, and there is not a closed-form for this term. Since the entropy $\mathbb{H}[q(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{s}^t, \mathbf{A}^0)]$ is a constant, we only optimize the cross entropy term in $\mathcal{L}_{\text{edge}}(t)$ via Monte Carlo estimates. We leave the work of variance reduction to the future.

For the node selection term $\mathcal{L}_{\text{node}}(t)$, we show that $q(\mathbf{s}^t|\mathbf{A}^t, \mathbf{A}^0)$ has closed-form expression. In particular, we first derive the posterior of the node degree distribution $q(\mathbf{d}^t|\mathbf{A}^t, \mathbf{A}^0)$ as follows:

$$q(\mathbf{d}^{t-1}|\mathbf{A}^t, \mathbf{A}^0) = q(\mathbf{d}^{t-1}|\mathbf{d}^t, \mathbf{d}^0) = \prod_{i=1}^{N} q(\mathbf{d}_i^{t-1}|\mathbf{d}_i^t, \mathbf{d}_i^0),$$

$$\text{where } q(\mathbf{d}_i^{t-1}|\mathbf{d}_i^t, \mathbf{d}_i^0) = \text{Bin}(k = \Delta_i^t, n = \Delta_i^0, p = \gamma_t),$$

$$\text{with } \Delta_i^t = \mathbf{d}_i^{t-1} - \mathbf{d}_i^t,\ \Delta_i^0 = \mathbf{d}_i^0 - \mathbf{d}_i^t,\ \gamma_t = \frac{\beta_t \bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}. \tag{8}$$

Here $\text{Bin}(k; n, p)$ is a binomial distribution parameterized by n and p. Intuitively, a node degree $\mathbf{d}_i^{t-1}$ is only relevant to the node's degrees $\mathbf{d}_i^0$ and $\mathbf{d}_i^t$ at steps 0 and t. The actual edges do not affect the degree probability since each edge is added or removed independently. We provide formal proof and discuss the forward node degree distribution in Appendix A.2.

Since $\mathbf{s}_i^t = \mathbb{1}[\mathbf{d}_i^{t-1} \neq \mathbf{d}_i^t]$, we can compute the probability $q(\mathbf{s}_i^t = 1|\mathbf{d}_i^t, \mathbf{d}_i^0)$, which is $1 - q(\mathbf{d}_i^{t-1} = \mathbf{d}_i^t|\mathbf{d}_i^t, \mathbf{d}_i^0)$. Finally, we obtain the closed-form posterior:

$$q(\mathbf{s}^t|\mathbf{d}^t, \mathbf{d}^0) = \prod_{i=1}^{N} q(\mathbf{s}_i^t|\mathbf{d}_i^t, \mathbf{d}_i^0), \text{ where} \tag{9}$$

$$q(\mathbf{s}_i^t|\mathbf{d}_i^t, \mathbf{d}_i^0) = \mathcal{B}\big(\mathbf{s}_i^t; 1 - (1-\gamma_t)^{\Delta_i^0}\big).$$

The KL divergence $\mathcal{L}_{\text{node}}(t)$ turns out to be comparisons between Bernoulli distributions.

3.4. Degree-guided graph generation

A graph's node degrees are often strongly correlated to its other statistics, so it is important for a generative model to capture the node degrees of training graphs. Here we directly incorporate degree information in the proposed generative model.

We explicitly model node degrees $\mathbf{d}^0$ of a graph $\mathbf{A}^0$ as a random variable, then the forward process becomes

$$q(\mathbf{A}^{1:T}|\mathbf{A}^0) = q(\mathbf{A}^{1:T}|\mathbf{A}^0)q(\mathbf{d}^0|\mathbf{A}^0). \tag{10}$$

Here $q(\mathbf{d}^0|\mathbf{A}^0) = 1$ because $\mathbf{d}^0$ is determined by $\mathbf{A}^0$. We also include $\mathbf{d}^0$ into the generative model $p(\mathbf{A}^0, \mathbf{d}^0)$. If the model guarantees that $\mathbf{d}^0$ is the node degrees of $\mathbf{A}^0$, then $p_\theta(\mathbf{A}^0) = p_\theta(\mathbf{A}^0, \mathbf{d}^0)$ still models graph data $\mathbf{A}^0$. Even if $p_\theta(\mathbf{A}^0, \mathbf{d}^0)$ allows $\mathbf{d}^0$ to differ a little from the true node degrees, it is still valid to maximize the likelihood $p_\theta(\mathbf{A}^0, \mathbf{d}^0 = \mathbf{A}^0\mathbf{1})$ because model training will encourage the model to generate $\mathbf{A}^0$ and $\mathbf{d}^0$ to match each other. We decompose the model by:

$$p_\theta(\mathbf{A}^0, \mathbf{d}^0) = p_\theta(\mathbf{d}^0)p_\theta(\mathbf{A}^0|\mathbf{d}^0). \tag{11}$$

With this decomposition, we first sample arbitrary node degrees $\mathbf{d}^0$ from $p_\theta(\mathbf{d}^0)$, then generate a graph with the degree constraint (See Alg. 1). Correspondingly, the denoising model becomes

$$p_\theta(\mathbf{A}^{0:T}, \mathbf{s}^{1:T}, \mathbf{d}^0) = p_\theta(\mathbf{d}^0) p_\theta(\mathbf{A}^{0:T}, \mathbf{s}^{1:T} | \mathbf{d}^0). \quad (12)$$

We separate the optimizations for the node degree model $p_\theta(\mathbf{d}^0)$ and the graph denoising model $p_\theta(\mathbf{A}^{0:T}, \mathbf{s}^{1:T}|\mathbf{d}^0)$. The entire training objective is

$$\mathcal{L}(\mathbf{A}^0, \mathbf{d}^0; \theta) = \mathbb{E}_q \Bigg[\underbrace{\log p_\theta(\mathbf{d}^0)}_{\mathcal{L}(\mathbf{d}^0;\theta)} + \underbrace{\log \frac{p_\theta(\mathbf{A}^{0:T}, \mathbf{s}^{1:T}|\mathbf{d}^0)}{q(\mathbf{A}^{1:T}, \mathbf{s}^{1:T}|\mathbf{A}^0)}}_{\mathcal{L}(\mathbf{A}^0|\mathbf{d}^0;\theta)} \Bigg].$$

(See Appendix B.2 for detailed derivation.) For $\mathcal{L}(\mathbf{d}^0; \theta)$, we treat the learning of node degree distribution as a sequence modeling task. The decomposition of $\mathcal{L}(\mathbf{A}^0|\mathbf{d}^0; \theta)$ remains the same as Eqn. (7), except that all terms related to the graph denoising model are now conditioning on $\mathbf{d}^0$. In particular, for the node selection distribution, we consider a special parameterization by setting $p_\theta(\mathbf{s}^t|\mathbf{A}^t, \mathbf{d}^0) := q(\mathbf{s}^t|\mathbf{d}^t, \mathbf{d}^0)$ in Eqn. (9). Note that now the node selection distribution contains no learnable parameters. Moreover, since the KL divergence $\mathcal{L}_{\text{node}}(t)$ is now zero, we can further simplify the $\mathcal{L}(\mathbf{A}^0|\mathbf{d}^0; \theta)$ into

$$\mathcal{L}(\mathbf{A}^0|\mathbf{d}^0;\theta) = \mathbb{E}_q \Bigg[\log \frac{p(\mathbf{A}^T)}{q(\mathbf{A}^T|\mathbf{A}^0)} + \log p_\theta(\mathbf{A}^0|\mathbf{A}^1, \mathbf{s}^1, \mathbf{d}^0) + \sum_{t=2}^{T} \log \frac{p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{s}^t, \mathbf{d}^0)}{q(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{s}^t, \mathbf{A}^0)} \Bigg]. \quad (13)$$

In our framework, the node degree constraint $\mathbf{d}^0$ is mainly imposed on $p_\theta(\mathbf{s}^t|\mathbf{A}^t, \mathbf{d}^0)$ – only nodes with a degree below the specified degree $\mathbf{d}^0$ may be selected to participate in the edge prediction. On the other hand, though the exact probability $p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{s}^t, \mathbf{d}^0)$ includes information about the maximum number of edges ($\mathbf{d}^0 - \mathbf{d}^t$) that can be added to nodes, this can be not easy to track during the edge formation. Here we consider simply augmenting the inputs to the neural network with $\mathbf{d}^0$. In practice, we found that the specified node degrees $\mathbf{d}^0$ can accurately control the actual node degrees of the generated graphs.

Degree-guided generation turns out to be very useful in modeling large graphs. We reason that the $\mathbf{d}^0$ significantly reduces the possible trajectories a graph can evolve along, thus reducing the modeling complexity.

3.5. Implementation

We briefly describe the implementation of $p_\theta(\mathbf{s}^t|\mathbf{A}^t)$, $p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{s}^t)$, and $p_\theta(\mathbf{d}^0)$. Note we use the same network architecture for $p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{s}^t)$ and $p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{s}^t, \mathbf{d}^0)$, except the inputs to the latter includes $\mathbf{d}^0$. We treat $p_\theta(\mathbf{s}^t|\mathbf{A}^t)$ as a node classification problem and $p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{s}^t)$ as an link prediction problem. Both components share the same MPNN that takes $\mathbf{A}^t$ as the input and computes node representations $\mathbf{Z}^t \in \mathbb{R}^{N \times d_{\text{h}}}$ for all nodes. The hidden dimension d_{h} is a hyper-parameter here. Then a network head uses $\mathbf{Z}^t$ to predict $\mathbf{s}^t$, and another one uses $\mathbf{Z}^t[\mathbf{s}^t]$ to predict links between active nodes indicated by $\mathbf{s}^t$. For the node degree model $p_\theta(\mathbf{d}^0)$, if there are multiple graphs in the dataset, we use a recurrent neural network (RNN) to fit the histogram of node degrees. If there is only one graph with node degrees $\mathbf{d}^0$, then we set $p_\theta(\mathbf{d}^0) = 1$ directly. Implementation details are in Appendix C.

Algorithm 1 Degree-guided graph generation

Input: Empty graph $\mathbf{A}^T$, graph model $p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{s}^t)$, degree sequence model $p_\theta(\mathbf{d}^0)$, and diffusion steps T.
Output: Generated graph $\mathbf{A}^0$
Draw $\mathbf{d}^0 \sim p_\theta(\mathbf{d}^0)$
for $t = T, \ldots, 1$ **do**
  Draw $\mathbf{s}^t \sim q(\mathbf{s}^t|\deg(\mathbf{A}^t), \mathbf{d}^0)$.
  Draw $\mathbf{A}^{t-1} \sim p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{s}^t)$.
end for

3.6. Model analysis

Complexity analysis. Let integer M represent the number of edges in a graph, and K be the maximum number of active nodes during the reverse process. In each generation step t, the MPNN needs $O(M)$ operations to compute node representations, $O(N)$ operations to predict $\mathbf{s}^t$, and $O(K^2)$ operations to predict links between K active nodes. The factor K is relevant to noise scheduling: we find that K is smaller than N by at least one order of magnitude when the noise scheduling is linear. In a total of T generation steps, the overall running time $O\big(T \max(K^2, M)\big)$. As a comparison, previous diffusion-based models need running time $O(TN^2)$ because they need to make $O(N^2)$ link predictions at each time step.

Expressivity analysis. EDGE modifies a graph for multiple iterations to generate a sample. In each iteration, it adds new edges to the graph based on the graph structure in the prior iteration. Therefore, EDGE is NOT an edge-independent model and does not have the limitation analyzed by Chanpuriya et al. (2021), thus it has a theoretical advantage over previous one-shot generative models.

The ability of EDGE might be affected by the underlying MPNN, which may not be able to distinguish different graph structures due to expressivity issues (Xu et al., 2018). However, this issue can be overcome by choosing more expressive GNNs (Sato, 2020). We defer such discussion to future work.

4. Related Work

Edge-independent models, which assume edges are formed independently with some probabilities, are prevalent in probabilistic models for large networks. These models include classical models such as ER graph models (Erdos et al., 1960), SBMs (Holland et al., 1983), and recent neural models such as variational graph auto-encoders (Kipf & Welling, 2016b; Mehta et al., 2019; Li et al., 2020; Chen et al., 2022a), NetGAN and its variant (Bojchevski et al., 2018; Rendsburg et al., 2020). Recent works show that these models can not reproduce desiring statistics of the target network, such as triangle counts, clustering coefficient, and square counts (Seshadhri et al., 2020; Chanpuriya et al., 2021).

Deep auto-regressive (AR) graph models (Li et al., 2018; You et al., 2018; Liao et al., 2019; Zang & Wang, 2020; Han et al., 2023) generate graph edges by sequentially filling up an adjacency matrix to generate a graph. These algorithms are slow because they need to make N^2 predictions. Dai et al. (2020) proposes a method to leverage graph sparsity and predict only non-zero entries in the adjacency matrix. Long-term memory is a typical issue of these sequential models, so it is hard for them to model global graph properties. Moreover, these models are not invariant with respect to node orders of training graphs, and special techniques (Chen et al., 2021; Han et al., 2023) are often needed to train these models.

Diffusion-based generative models are shown to be powerful in generating high-quality graphs (Niu et al., 2020; Liu et al., 2019; Jo et al., 2022; Haefeli et al., 2022; Chen et al., 2022b; Vignac et al., 2022; Kong et al.). By "tailoring" a graph with multiple steps, these models can model edge correlations. They overcome the limitations of auto-regressive modes as well. However, all previous diffusion-based models focus on generation tasks with small graphs. This work aims to scale diffusion-based models to large graphs.

5. Experiments

We empirically evaluate our proposed approach from two perspectives: whether it can capture statistics of training graphs and whether it can generate graphs efficiently.

5.1. Experimental setup

Datasets. We conduct experiments on both generic graph datasets and large networks. The generic graph datasets consist of multiple graphs of varying sizes. Here we consider Community and Ego datasets (You et al., 2018), all of which contain graphs with hundreds of nodes. We also consider four real-world networks, Polblogs (Adamic & Glance, 2005), Cora (Sen et al., 2008), Road-Minnesota (Rossi & Ahmed, 2015), and PPI (Stark et al., 2010). Each of these networks contains thousands of nodes. We also use the QM9 dataset (Ramakrishnan et al., 2014) to demonstrate that EDGE can be easily extended to generate graphs with attributes. The statistics of the datasets are shown in Table 1.

	#nodes	#edges	#graphs	feature
Community	[60, 160]	[231, 1,965]	510	
Ego	[50, 399]	[57, 1,071]	757	
QM9	[1,9]	[0, 28]	133,885	✓
Polblogs	1,222	16,714	1	
Cora	2,485	5,069	1	
Road-MN	2,640	6,604	1	
PPI	3,852	37,841	1	

Table 1. Dataset statistics

Baselines. For generic graphs, We compare EDGE to six recent deep generative graph models, which include two auto-regressive graph models, GraphRNN (You et al., 2018) and GRAN (Liao et al., 2019), three diffusion-based models, GDSS (Jo et al., 2022), DiscDDPM (Haefeli et al., 2022) and DiGress (Vignac et al., 2022), and one flow-based model, GraphCNF (Lippe & Gavves, 2020). For large networks, we follow Chanpuriya et al. (2021) and use six edge-independent models, which include VGAE (Kipf & Welling, 2016b), CELL (Rendsburg et al., 2020), TSVD (Seshadhri et al., 2020), and three methods proposed by Chanpuriya et al. (2021) (CCOP, HDOP, Linear). We also include GraphRNN as a baseline because it is still affordable to train it on large networks. For the QM9 dataset, We compare EDGE against GDSS (Jo et al., 2022) and DiGress (Vignac et al., 2022). The implementation of our model is available at github.com/tufts-ml/graph-generation-EDGE.

Evaluation. We examine the generated generic graphs with both structure-based and neural-based metrics. For structured-based metrics, we evaluate the Maximum Mean Discrepancy (MMD) (Gretton et al., 2012) between test graphs and generated graphs in terms of degrees, clustering coefficients, and orbit counts (You et al., 2018). For neural-based metrics, we evaluate the FID and the MMD RBF metrics proposed by Thompson et al. (2022). All implementations of the evaluation are provided by Thompson et al. (2022). For all these metrics, the smaller, the better.

For each large network, we follow Chanpuriya et al. (2021) and evaluate how well the graph statistics of the generated network can match ground truths, which are statistics computed from training data. We consider the following statistics: power-law exponent of the degree sequence (PLE); normalized triangle counts (NTC); global clustering coefficient (CC) (Chanpuriya et al., 2021); characteristic path length (CPL); and assortativity coefficient (AC) (Newman, 2002). We also report the edge overlap ratio (EO) between the generated network and the original one to check to which degree a model memorizes the graph. A graph generated by a good model should have statistics similar to true values

	Community					Ego				
	Structure-based (MMD)			Neural-based		Structure-based (MMD)			Neural-based	
	Deg.	Clus.	Orb.	FID	RBF MMD	Deg.	Clus.	Orb.	FID	RBF MMD
GRNN	0.1440	0.0535	**0.0198**	8.3869	0.1591	0.0768	1.1456	0.1087	90.5655	0.6827
GRAN	0.1022	0.0894	**0.0198**	64.1145	0.0749	0.5778	0.3360	**0.0406**	489.9598	0.2633
GraphCNF	0.1129	1.2882	**0.0197**	29.1526	0.1341	0.1010	0.7654	0.0820	18.7929	0.0896
GDSS	0.0535	0.2072	**0.0196**	6.5531	0.0443	0.8189	0.6032	0.3315	60.6100	0.4331
DiscDDPM	0.1238	0.6549	0.0246	8.6321	0.0840	0.4613	0.1681	0.0633	42.7994	0.1561
DiGress	0.0409	**0.0167**	0.0298	3.4261	0.0460	0.0708	**0.0092**	0.1205	18.6794	**0.0489**
EDGE	**0.0175**	0.0689	**0.0198**	**2.2378**	**0.0227**	**0.0579**	0.1773	**0.0519**	**15.7614**	**0.0658**

Table 2. Generation performance on generic graphs. We used unpaired t-tests to compare the results; the numbers in bold indicate the method is better at the 5% significance level, and the second-best method is underlined. We provide standard deviation in Appendix F.

computed from the training graph. At the same time, it should have a small EO with the training network, which means that the model should not simply memorize the input data.

For the QM9 dataset, we evaluate the Validity, Uniqueness, Fréchet ChemNet Distance (Preuer et al., 2018) and Scaffold similarity (Bemis & Murcko, 1996) on the samples generated from baselines and our proposed method. We use molsets library (Polykovskiy et al., 2020) to implement the evaluation.

5.2. Evaluation of sample quality

Generic graph generation. Table 2 summarizes the evaluation of generated graphs on the Community and Ego datasets. Best performances are in bold, and second-best performances are underscored. EDGE outperforms all baselines on 8 out of 10 metrics. For the other two metrics, EDGE only performs slightly worse than the best. We hypothesize that EDGE gains advantages by modeling node degrees because they are informative to the graph structure.

Large network generation. Unlike edge-independent models, the edge overlap ratios in the GraphRNN and our approach are not tunable. To make a fair comparison, we report the performance of the edge-independent models that have a similar or higher EO than GraphRNN and EDGE. Table 3 shows the statistics of the network itself (labeled as "True") and statistics computed from generated graphs. The statistics nearest to true values are considered as best performances, which are in bold. Second-best performances are underscored.

The proposed approach shows superior performances on all four networks. The improvements on Polblogs and PPI networks are clear. On the Road-Minnesota dataset, EDGE has a much smaller EO than edge-independent models, but its performances in terms of capturing graph statistics are similar to those models. On the Cora dataset, EDGE also has an EO much smaller than edge-independent models, but

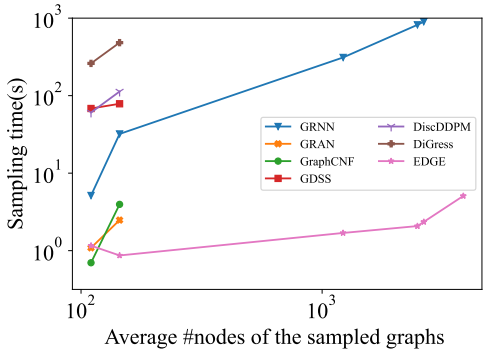


Figure 3. Sampling speed comparison over different models.

it slightly improves over these models. Road-Minnesota and Cora networks are both sparse networks – the message-passing neural model may not work at its full strength. We notice that GraphRNN can not even compete with edge-independent models. We also visualize the generated graphs of Polblogs in Figure 4.

5.3. Efficiency

We compare the sampling efficiency of EDGE against other deep generative graph models. We record the average time for a model to sample one graph to make a consistent comparison over all datasets. The average sampling time for each dataset is averaged over 128 runs. Figure 3 shows the relationship between sampling time and graph sizes. Except for GraphRNN, all baseline neural models can only generate graphs for Community and Ego datasets, which contain 110 and 144 nodes on average. Our approach runs only slower than GraphCNF on the Community dataset by 0.5s. On large graphs, our model has a clear advantage in terms of running time. Note that our model spends less time on an Ego graph than a Community graph, though an Ego graph, on average, contains more nodes than a Community graph. This is because the computation of our model scales with the number of edges, and Ego graphs are often sparser than Community graphs.

	Polblogs						Cora					
	EO	PLE	NTC	CC	CPL	AC	EO	PLE	NTC	CC	CPL	AC
True	100	1.414	1	0.226	2.738	-0.221	100	1.885	1	0.090	6.311	-0.071
OPB	24.5	1.395	0.667	0.150	2.524	-0.143	10.9	**1.852**	0.097	0.008	4.476	-0.037
HDOP	16.4	1.393	0.687	0.153	2.522	-0.131	0.9	**1.849**	0.113	0.009	4.770	-0.030
CELL	26.8	1.385	0.810	0.211	2.534	-0.230	10.3	1.774	0.009	0.002	5.799	-0.018
CO	20.1	1.975	0.045	0.028	2.502	0.068	9.7	1.776	0.009	0.002	5.653	0.010
TSVD	32.0	1.373	0.872	0.205	2.532	**-0.216**	6.7	**1.858**	0.349	0.028	4.908	-0.006
VGAE	3.6	1.723	0.05	0.001	2.531	-0.086	1.5	1.717	0.120	0.220	4.934	0.002
GRNN	9.6	1.333	0.354	0.095	2.566	0.096	0.4	1.822	0.043	0.011	**6.146**	0.043
EDGE	16.5	**1.398**	**0.977**	**0.217**	**2.647**	**-0.214**	1.1	1.755	**0.446**	**0.034**	4.995	**-0.046**

	Road-Minnesota						PPI					
	EO	PLE	NTC	CC	CPL	AC	EO	PLE	NTC	CC	CPL	AC
True	100	2.147	1	0.028	35.349	-0.187	100	1.462	1	0.092	3.095	-0.099
OPB	29.7	**2.188**	0.083	0.002	8.036	0.009	16.3	1.443	0.640	0.058	2.914	**-0.089**
HDOP	13.2	**2.192**	0.208	0.004	8.274	-0.024	6.9	1.444	0.638	0.058	2.917	-0.086
CELL	30.7	2.267	0.053	0.001	10.219	**-0.082**	6.7	1.400	0.248	0.040	**3.108**	0.176
CO	19.8	2.044	2.845	**0.040**	11.478	-0.012	9.9	1.754	0.015	0.006	3.046	0.043
TSVD	19.4	**2.172**	0.060	0.001	8.431	0.006	13.2	1.426	0.848	0.077	2.867	**-0.089**
VGAE	1.3	1.678	0.096	0.009	11.120	-0.027	0.5	1.362	0.091	0.012	2.991	0.054
GRNN	0.6	1.570	0.099	0.007	**11.695**	0.006	OOM	OOM	OOM	OOM	OOM	OOM
EDGE	0.8	1.910	**0.962**	0.011	9.125	-0.063	7.5	**1.449**	**0.981**	**0.091**	3.028	**-0.107**

Table 3. Graph statistics of generated large networks. EDGE generates graphs with statistics that are much closer to the ground truths.

	Validity↑	Uniqueness↑	FCD↓	Scaf. Sim.↑
GDSS	95.7	98.5	2.9	-
DiGress	99.0	**100**	**0.151**	**0.908**
EDGE	**99.1**	**100**	0.458	0.763

Table 4. Generative performance on the QM9 dataset

	$\mathbf{s}^{1:T}$	$\mathbf{d}^0$	PLE	NTC	CC	CPL	AC	Speed
True			1.414	1	0.226	2.738	-0.221	
G(N,0.5)			OOM	OOM	OOM	OOM	OOM	OOM
G(N,0)			1.341	3.234	0.237	**2.747**	-0.304	15.3s
G(N,0)	✓		1.383	2.364	0.251	2.638	-0.331	2.1s
G(N,0)	✓	✓	**1.398**	**0.977**	**0.217**	2.647	**-0.214**	**1.7s**

Table 5. Performance of EDGE's variants on the Polblogs dataset.

5.4. Generative performance on QM9 dataset

We further investigate EDGE's ability of generated graphs with node and edge attributes. To include node attributes, we first extend the basic EDGE model with a hierarchical generation process that can also sample node attributes. We put the details of this extension in Appendix E. We evaluate the extended EDGE model on the QM9 dataset and compare it with other neural baselines. The results in Table 4 show that the extended EDGE model has a performance comparable with that of DiGress. Note that DiGress is specially designed for molecule generation, and our model runs much faster than DiGress.

5.5. Ablation studies

Diffusion variants. The random variables $\mathbf{s}^{1:T}$ and $\mathbf{d}^0$ play important roles in EDGE's good performances, and we verify that through an ablation study on the Polblogs dataset. We use four diffusion configurations: 1) setting $G(N, 0.5)$ as the convergent distribution and directly using an MPNN as the denoising model $p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t)$; 2) setting $G(N, 0)$ as the convergent distribution and directly using an MPNN as the denoising model (without modeling active nodes and degree guidance); 3) the EDGE model without degree guidance, and 4) the EDGE model. Table 5 shows the performances of the four models. If we set the convergent distribution to $G(N, 0.5)$, we can not even train such as model since it requires an excessively large amount of GPU memory. This justifies our use of $G(N, 0)$ as the convergent distribution. The introduction of $\mathbf{s}^{1:T}$ (Section 3.2) significantly improves the sampling speed. Finally, the EDGE approach, which explicitly models node degrees $\mathbf{d}^0$ and generates graphs with degree guidance, further improves the generative performance.

Diffusion steps vs. model performance. In EDGE, the number of diffusion steps T decides how many nodes would actively participate in the edge prediction. Here we investigate how it affects the model performance under linear

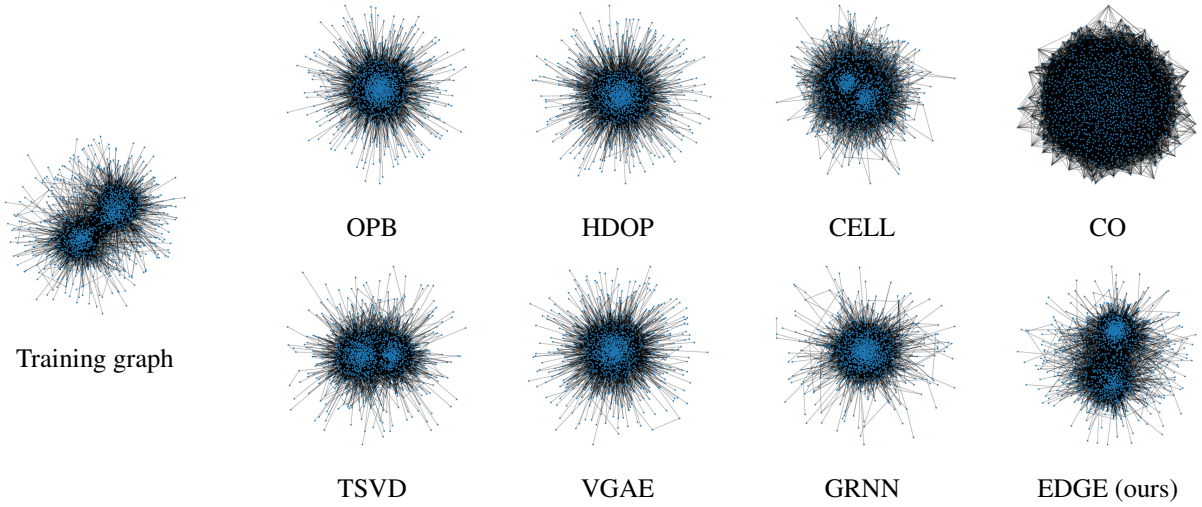


Figure 4. Visualization of samples for the Polblogs dataset. We observe that only CELL, TSVD, and EDGE can learn the basic structure of the ground-truth network, while other baselines fail. The network sampled from EDGE appears to be more similar to the training graph.

noise scheduling.

		EO	PLE	NTC	CC	CPL	AC
Ploblogs	True	100	1.414	1	0.226	2.738	-0.221
	64	1.8	1.380	1.148	0.235	2.800	-0.202
	128*	14.9	1.386	1.030	0.238	2.747	-0.238
	256*	16.5	1.398	0.977	0.217	2.647	-0.214
	512*	15.0	1.398	0.923	0.218	2.635	-0.268
	1024*	16.5	1.400	0.991	0.219	2.665	-0.246
Cora	True	100	1.885	1	0.090	6.311	-0.071
	64*	0.9	1.755	0.446	0.034	4.995	-0.046
	128	1.1	1.747	0.555	0.042	5.017	-0.050
	256	0.8	1.753	0.360	0.027	4.818	-0.041
	512	0.8	1.753	0.360	0.027	4.818	-0.042
	1024	0.9	1.762	0.348	0.027	4.778	-0.034
Road-MN	True	100	2.147	1	0.028	35.349	-0.187
	64*	0.8	1.910	0.962	0.011	9.125	-0.063
	128	1.2	1.803	1.232	0.041	6.501	-0.030
	256	0.8	1.953	1.057	0.014	7.471	-0.005
	512	1.3	1.965	1.472	0.020	7.710	-0.006
	1024	1.2	1.983	2.491	0.035	7.906	-0.034
PPI	True	100	1.462	1	0.092	3.095	-0.099
	64	7.4	1.421	2.455	-0.116	3.498	-0.116
	128	6.2	1.419	1.503	0.126	3.384	-0.147
	256*	7.5	1.449	0.981	0.091	3.028	-0.107
	512*	7.0	1.438	1.101	0.099	3.244	-0.107
	1024*	7.1	1.441	0.925	0.074	3.150	-0.101

Table 6. Large diffusion steps T does not necessarily improve model performance. Good diffusion steps are labeled with "*".

Specifically, we train our model on three large networks with $T \in \{64, 128, 256, 512, 1024\}$ and report the model performance in Table 6. Unlike traditional diffusion models in which more diffusion steps usually yield better performance, a large T for our model does not always improve the performance. For instance, $T = 64$ gives the best performance in the Cora and Road-Minnesota datasets. Our explanation for this observation is the high level of sparsity in training graphs. If we have a large T, the total number of generation steps, the model can only identify a few active nodes and predict edges between them in each time step. The model faces a highly imbalanced classification problem, which may lead to poor model convergence. Such an issue is not observed for relatively denser graphs, e.g. Polblogs and PPI datasets, which require a relatively large T to guarantee good model performances. When T is large enough ($T = 128$ for Polbogs and $T = 256$ for PPI), further increasing T does not improve the model performance.

6. Conclusion

In this work, we propose EDGE, a generative graph model based on a discrete diffusion process. By leveraging the sparsity in the diffusion process, EDGE significantly improves the computation efficiency and scales to graphs with thousands of nodes. By explicitly modeling node degrees, EDGE improves its ability in capturing important statistics of training graphs. Our extensive empirical study shows that EDGE has superior performance in benchmark graph generation in terms of both computational efficiency and generation quality.

Acknowledgment

We thank anonymous reviewers for their valuable feedback. Xiaohui Chen and Li-Ping Liu are partially supported by the National Science Foundation under Grant No. 2239869.

References

Adamic, L. A. and Glance, N. The political blogosphere and the 2004 us election: divided they blog. In *Proceedings of the 3rd international workshop on Link discovery*, pp. 36–43, 2005.

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.

Bemis, G. W. and Murcko, M. A. The properties of known drugs. 1. molecular frameworks. *Journal of medicinal chemistry*, 39(15):2887–2893, 1996.

Bojchevski, A., Shchur, O., Zügner, D., and Günnemann, S. Netgan: Generating graphs via random walks. In *International conference on machine learning*, pp. 610–619. PMLR, 2018.

Chakrabarti, D. and Faloutsos, C. Graph mining: Laws, generators, and algorithms. *ACM computing surveys (CSUR)*, 38(1):2–es, 2006.

Chanpuriya, S., Musco, C., Sotiropoulos, K., and Tsourakakis, C. On the power of edge independent graph models. *Advances in Neural Information Processing Systems*, 34:24418–24429, 2021.

Chen, X., Han, X., Hu, J., Ruiz, F. J., and Liu, L. Order matters: Probabilistic modeling of node sequence for graph generation. *arXiv preprint arXiv:2106.06189*, 2021.

Chen, X., Chen, X., and Liu, L. Interpretable node representation with attribute decoding. *arXiv preprint arXiv:2212.01682*, 2022a.

Chen, X., Li, Y., Zhang, A., and Liu, L.-p. Nvdiff: Graph generation through the diffusion of node vectors. *arXiv preprint arXiv:2211.10794*, 2022b.

Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

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.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Dhariwal, P. and Nichol, A. Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems*, 34:8780–8794, 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.

Elfwing, S., Uchibe, E., and Doya, K. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018.

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

Fey, M. and Lenssen, J. E. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., and Smola, A. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012.

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.

Han, X., Chen, X., Ruiz, F. J., and Liu, L.-P. Fitting autoregressive graph generative models through maximum likelihood estimation. *Journal of Machine Learning Research*, 24(97):1–30, 2023.

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

Holland, P. W., Laskey, K. B., and Leinhardt, S. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.

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.

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.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016a.

Kipf, T. N. and Welling, M. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016b.

Kong, L., Cui, J., Sun, H., Zhuang, Y., Prakash, B. A., and Zhang, C. Autoregressive diffusion model for graph generation.

Li, J., Yu, J., Li, J., Zhang, H., Zhao, K., Rong, Y., Cheng, H., and Huang, J. Dirichlet graph variational autoencoder. *Advances in Neural Information Processing Systems*, 33: 5274–5283, 2020.

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. In *Advances in Neural Information Processing Systems*, pp. 4255–4265, 2019.

Lippe, P. and Gavves, E. Categorical normalizing flows via continuous transformations. *arXiv preprint arXiv:2006.09790*, 2020.

Liu, J., Kumar, A., Ba, J., Kiros, J., and Swersky, K. Graph normalizing flows. *Advances in Neural Information Processing Systems*, 32, 2019.

Lusher, D., Koskinen, J., and Robins, G. *Exponential random graph models for social networks: Theory, methods, and applications*. Cambridge University Press, 2013.

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

Mehta, N., Duke, L. C., and Rai, P. Stochastic blockmodels meet graph neural networks. In *International Conference on Machine Learning*, pp. 4466–4474. PMLR, 2019.

Newman, M. E. Assortative mixing in networks. *Physical review letters*, 89(20):208701, 2002.

Newman, M. E., Watts, D. J., and Strogatz, S. H. Random graph models of social networks. *Proceedings of the national academy of sciences*, 99(suppl_1):2566–2572, 2002.

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.

O'Bray, L., Horn, M., Rieck, B., and Borgwardt, K. Evaluation metrics for graph generative models: Problems, pitfalls, and practical solutions. *arXiv preprint arXiv:2106.01098*, 2021.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Polykovskiy, D., Zhebrak, A., Sanchez-Lengeling, B., Golovanov, S., Tatanov, O., Belyaev, S., Kurbanov, R., Artamonov, A., Aladinskiy, V., Veselov, M., et al. Molecular sets (moses): a benchmarking platform for molecular generation models. *Frontiers in pharmacology*, 11:565644, 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.

Rendsburg, L., Heidrich, H., and Von Luxburg, U. Netgan without gan: From random walks to low-rank approximations. In *International Conference on Machine Learning*, pp. 8073–8082. PMLR, 2020.

Rossi, R. and Ahmed, N. The network data repository with interactive graph analytics and visualization. In *Proceedings of the AAAI conference on artificial intelligence*, volume 29, 2015.

Sato, R. A survey on the expressive power of graph neural networks. *arXiv preprint arXiv:2003.04078*, 2020.

Schuster, M. and Paliwal, K. K. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

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.

Seshadhri, C., Sharma, A., Stolman, A., and Goel, A. The impossibility of low-rank representations for triangle-rich complex networks. *Proceedings of the National Academy of Sciences*, 117(11):5631–5637, 2020.

Shi, Y., Huang, Z., Feng, S., Zhong, H., Wang, W., and Sun, Y. Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509*, 2020.

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.

Stark, C., Breitkreutz, B.-J., Chatr-Aryamontri, A., Boucher, L., Oughtred, R., Livstone, M. S., Nixon, J., Van Auken, K., Wang, X., Shi, X., et al. The biogrid interaction database: 2011 update. *Nucleic acids research*, 39 (suppl_1):D698–D704, 2010.

Thompson, R., Knyazev, B., Ghalebi, E., Kim, J., and Taylor, G. W. On evaluation metrics for graph generative models. *arXiv preprint arXiv:2201.09871*, 2022.

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.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2018.

Ying, X. and Wu, X. Graph generation with prescribed feature constraints. In *Proceedings of the 2009 SIAM International Conference on Data Mining*, pp. 966–977. SIAM, 2009.

You, J., Ying, R., Ren, X., Hamilton, W. L., and Leskovec, J. GraphRNN: Generating realistic graphs with deep auto-regressive models. *arXiv preprint arXiv:1802.08773*, 2018.

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.

Appendix

A. Derivation of the $G(N, 0)$ Diffusion Process and the Degree Change Distribution

We first derive the forward and reverse transition distribution of the $G(N, 0)$ diffusion process, then we examine the distribution of node degree changes for both directions.

A.1. The $G(N, 0)$ diffusion process

We consider modeling the upper triangle of the adjacency matrix $\mathbf{A}^0$. Since we have $p = 0$ in our framework, for the forward transition kernel, we have

$$q(\mathbf{A}^t|\mathbf{A}^{t-1}) = \prod_{i,j:i<j} q(\mathbf{A}^t_{i,j}|\mathbf{A}^{t-1}_{i,j}), \text{ with } q(\mathbf{A}^t_{i,j}|\mathbf{A}^{t-1}_{i,j}) = \mathcal{B}(\mathbf{A}^t_{i,j}; (1-\beta_t)\mathbf{A}^{t-1}_{i,j}). \tag{14}$$

$$q(\mathbf{A}^t|\mathbf{A}^0) = \prod_{i,j:i<j} q(\mathbf{A}^t_{i,j}|\mathbf{A}^0_{i,j}), \text{ with } q(\mathbf{A}^t_{i,j}|\mathbf{A}^0_{i,j}) = \mathcal{B}(\mathbf{A}^t_{i,j}; \bar{\alpha}_t\mathbf{A}^0_{i,j}). \tag{15}$$

The posterior $q(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{A}^0)$, whose form is discussed in Eqn. (3), is decomposed into

$$\begin{aligned} q(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{A}^0) &= \frac{q(\mathbf{A}^t|\mathbf{A}^{t-1})q(\mathbf{A}^{t-1}|\mathbf{A}^0)}{q(\mathbf{A}^t|\mathbf{A}^0)} \\ &= \prod_{i,j:i<j} \frac{q(\mathbf{A}^t_{i,j}|\mathbf{A}^{t-1}_{i,j})q(\mathbf{A}^{t-1}_{i,j}|\mathbf{A}^0_{i,j})}{q(\mathbf{A}^t_{i,j}|\mathbf{A}^0_{i,j})} \\ &= \prod_{i,j:i<j} q(\mathbf{A}^{t-1}_{i,j}|\mathbf{A}^t_{i,j}, \mathbf{A}^0_{i,j}) \end{aligned} \tag{16}$$

The entry-wise posterior distribution $q(\mathbf{A}^{t-1}_{i,j}|\mathbf{A}^t_{i,j}, \mathbf{A}^0_{i,j})$ is the key to deriving the distribution of active nodes. Here, we describe the detailed form of this distribution. For any value $p \in [0, 1]$, we have the form

$$q(\mathbf{A}^{t-1}_{i,j}|\mathbf{A}^t_{i,j}, \mathbf{A}^0_{i,j}) = \mathcal{B}(\mathbf{A}^{t-1}_{i,j}; \frac{p_1}{p_0 + p_1}), \text{where} \tag{17}$$

$$p_1 = [(1 - \beta_t + \beta_t p)\mathbf{A}^t_{i,j} + (\beta_t - \beta_t p)(1 - \mathbf{A}^t_{i,j})][\bar{\alpha}_{t-1}\mathbf{A}^0_{i,j} + (1 - \bar{\alpha}_{t-1})p] \tag{18}$$

$$p_0 = [(\beta_t p)\mathbf{A}^t_{i,j} + (1 - \beta_t p)(1 - \mathbf{A}^t_{i,j})][1 + \bar{\alpha}_{t-1}p - \bar{\alpha}_{t-1}\mathbf{A}^0_{i,j} - p] \tag{19}$$

Note that the posterior derived in Sohl-Dickstein et al. (2015); Hoogeboom et al. (2021) is only applicable to the case where $p = 0.5$, the above posterior is more general. In particular, for $p = 0$ in our case, the posterior can be simplified into the following three cases

$$q(\mathbf{A}^{t-1}_{i,j}|\mathbf{A}^t_{i,j}, \mathbf{A}^0_{i,j}) = \begin{cases} \mathcal{B}(\mathbf{A}^{t-1}_{i,j}; 0), & \mathbf{A}^0_{i,j} = 0 \\ \mathcal{B}(\mathbf{A}^{t-1}_{i,j}; 1), & \mathbf{A}^0_{i,j} = 1, \mathbf{A}^t_{i,j} = 1 \\ \mathcal{B}(\mathbf{A}^{t-1}_{i,j}; \frac{\beta_t \bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}), & \mathbf{A}^0_{i,j} = 1, \mathbf{A}^t_{i,j} = 0 \end{cases} \tag{20}$$

We provide an intuitive interpretation of the three cases. Since we are considering an edge-removing process, for the case where $\mathbf{A}^0_{i,j} = 0$, the probability an edge (i, j) is formed at timestep $t - 1$ is 0 (note that the event $(\mathbf{A}^0_{i,j} = 0, \mathbf{A}^t_{i,j} = 1)$ is unlikely to happen). The case where $\mathbf{A}^0_{i,j} = \mathbf{A}^t_{i,j} = 1$ indicates the edge (i, j) is not removed for all timesteps from 0 to t, therefore, $\mathbf{A}^{t-1}_{i,j}$ always equals to 1. The last case is the only case with uncertainty since an edge (i, j) can be removed at any timestep before t.

A.2. The distribution of active nodes

We now have the entry-wise forward distribution and posterior distribution. We can compute the probability that a node has a degree change at each time step. Here we first discuss the form of the forward and posterior degree distributions, which can be directly applied to calculate the degree change distributions:

Property 1. The forward degree distributions have the form

$$q(\mathbf{d}^t|\mathbf{d}^0) = \prod_{i=1}^{N} q(\mathbf{d}_i^t|\mathbf{d}_i^0), \text{ where } q(\mathbf{d}_i^t|\mathbf{d}_i^0) = \text{Binomial}(k = \mathbf{d}_i^t, n = \mathbf{d}_i^0, p = \bar{\alpha}_t). \tag{21}$$

$$q(\mathbf{d}^t|\mathbf{d}^{t-1}) = \prod_{i=1}^{N} q(\mathbf{d}_i^t|\mathbf{d}_i^{t-1}), \text{ where } q(\mathbf{d}_i^t|\mathbf{d}_i^{t-1}) = \text{Binomial}(k = \mathbf{d}_i^t, n = \mathbf{d}_i^{t-1}, p = 1 - \beta_t). \tag{22}$$

Intuitively, for $q(\mathbf{d}^t|\mathbf{d}^0)$, there are $\mathbf{d}_i^0$ edges connected to node i, each with probability $\bar{\alpha}_t$ to be kept at time step t. The probability the number of remaining edges equals $\mathbf{d}_i^t$ at time step t is a binomial distribution. A similar statement also holds for the one-step transition $q(\mathbf{d}^t|\mathbf{d}^{t-1})$, where an edge will have probability $1 - \beta_t$ be kept when transiting from $t - 1$ to t.

We also need to compute $q(\mathbf{d}^{t-1}|\mathbf{d}^t, \mathbf{d}^0)$, and we show that $q(\mathbf{d}^{t-1}|\mathbf{d}^t, \mathbf{d}^0) = q(\mathbf{d}^{t-1}|\mathbf{A}^t, \mathbf{A}^0)$. It holds because edges are removed independently. Since

$$\mathbf{d}_i^{t-1} = \sum_{j=1}^{N} \mathbf{A}_{i,j}^{t-1}, \tag{23}$$

and $\mathbf{A}_{i,j}^{t-1}$-s are independent variables. According to (20), $\mathbf{d}_i^{t-1}$ is the summation of three types of independent random variables: the first type is always 0, and the second type is always 1. We only need to consider the second and third types of variables, whose counts are respectively $\mathbf{d}_i^t$ and $(\mathbf{d}_i^0 - \mathbf{d}_i^t)$. Then $\mathbf{d}^{t-1}$ in $q(\mathbf{d}^{t-1}|\mathbf{A}^t, \mathbf{A}^0)$ is the sum of $\mathbf{d}_i^t$ and a random variable from $\text{Binomial}\left(n = \mathbf{d}_i^0 - \mathbf{d}_i^t, p = \frac{\beta_t \bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\right)$. It also indicates that $q(\mathbf{d}^{t-1}|\mathbf{d}^t, \mathbf{d}^0) = q(\mathbf{d}^{t-1}|\mathbf{A}^t, \mathbf{A}^0)$.

Property 2. The posterior degree distribution $q(\mathbf{d}^{t-1}|\mathbf{d}^t, \mathbf{d}^0)$ has the form:

$$q(\mathbf{d}^{t-1}|\mathbf{d}^t, \mathbf{d}^0) = \prod_{i=1}^{N} q(\mathbf{d}_i^{t-1}|\mathbf{d}_i^t, \mathbf{d}_i^0), \text{ with} \tag{24}$$

$$q(\mathbf{d}_i^{t-1}|\mathbf{d}_i^t, \mathbf{d}_i^0) = \text{Binomial}\left(k = (\mathbf{d}_i^{t-1} - \mathbf{d}_i^t); n = \mathbf{d}_i^0 - \mathbf{d}_i^t, p = \frac{\beta_t \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t}\right). \tag{25}$$

Now we have derived the forward and reverse degree distributions. It's obvious that when a node has no degree change from time step $t - 1$ to t or the other way around, we always have $\mathbf{d}_i^{t-1} = \mathbf{d}_i^t$. The probabilities of such events can be computed by querying the degree distributions $q(\mathbf{d}^t|\mathbf{d}^{t-1})$ or $q(\mathbf{d}^{t-1}|\mathbf{d}^t, \mathbf{d}^0)$. Let $\mathbf{s}_i^t$ be the random variable that node i has degree change at time step t. Below we show the forward and reverse degree change distributions:

Property 3. At timestep t, the forward degree change distribution for node i given $\mathbf{d}_i^{t-1}$ is

$$q(\mathbf{s}_i^t|\mathbf{d}_i^{t-1}) = \mathcal{B}\big(\mathbf{s}_i^t; 1 - (1 - \beta_t)^{\mathbf{d}_i^{t-1}}\big). \tag{26}$$

Property 4. At timestep t, the reverse degree change distribution for node i given $\mathbf{d}_i^t, \mathbf{d}_i^0$ is

$$q(\mathbf{s}_i^t|\mathbf{d}_i^t, \mathbf{d}_i^0) = \mathcal{B}\left(\mathbf{s}_i^t; 1 - \left(1 - \frac{\beta_t \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t}\right)^{\mathbf{d}_i^0 - \mathbf{d}_i^t}\right). \tag{27}$$

The distribution of active nodes from $q(\mathbf{s}_i^t|\mathbf{d}_i^{t-1})$ provides insightful supporting evidence that only a part of nodes may have degree change at each transition, motivating us the develop such a scalable generative framework. Controlling the number of nodes with degree change in the forward process can function as a principle to improve the noise scheduling algorithm. In practice, we only use the reverse degree change distribution when learning the reverse process. The reverse degree distribution is essential in improving the model expressivity since it enables graph generation with degree guidance.

B. Derivations of the Training Objectives

B.1. Derivation of the objective $\mathcal{L}(\mathbf{A}^0;\theta)$

To obtain the objective $\mathcal{L}(\mathbf{A}^0;\theta)$, we first need to derive the posterior of $\mathbf{A}^{t-1}$ that conditions on the introduced latent variables $\mathbf{s}^t$

$$
\begin{aligned}
q(\mathbf{A}^{t-1}|\mathbf{A}^t,\mathbf{s}^t,\mathbf{A}^0) &= \frac{q(\mathbf{A}^{t-1},\mathbf{A}^t,\mathbf{s}^t|\mathbf{A}^0)}{q(\mathbf{A}^t,\mathbf{s}^t|\mathbf{A}^0)} \\
&= \frac{q(\mathbf{A}^t|\mathbf{A}^{t-1},\mathbf{A}^0)q(\mathbf{s}^t|\mathbf{A}^t,\mathbf{A}^{t-1},\mathbf{A}^0)q(\mathbf{A}^{t-1}|\mathbf{A}^0)}{q(\mathbf{A}^t,\mathbf{s}^t|\mathbf{A}^0)} \\
&= \frac{q(\mathbf{A}^t|\mathbf{A}^{t-1})q(\mathbf{s}^t|\mathbf{A}^t,\mathbf{A}^{t-1})q(\mathbf{A}^{t-1}|\mathbf{A}^0)}{q(\mathbf{A}^t|\mathbf{A}^0)q(\mathbf{s}^t|\mathbf{A}^t,\mathbf{A}^0)}.
\end{aligned} \tag{28}
$$

By rearranging terms, we have

$$
q(\mathbf{A}^t|\mathbf{A}^{t-1})q(\mathbf{s}^t|\mathbf{A}^t,\mathbf{A}^{t-1}) = \frac{q(\mathbf{A}^{t-1}|\mathbf{A}^t,\mathbf{s}^t,\mathbf{A}^0)q(\mathbf{A}^t|\mathbf{A}^0)q(\mathbf{s}^t|\mathbf{A}^t,\mathbf{A}^0)}{q(\mathbf{A}^{t-1}|\mathbf{A}^0)}. \tag{29}
$$

The VLB $\mathcal{L}(\mathbf{A}^0;\theta)$ of $\log p(\mathbf{A}^0)$ is derived as follow

$$
\begin{aligned}
&\mathcal{L}(\mathbf{A}^0;\theta) \\
&= \mathbb{E}_q\big[\log \frac{p_\theta(\mathbf{A}^{0:T},\mathbf{s}^{1:T})}{q(\mathbf{A}^{1:T},\mathbf{s}^{1:T}|\mathbf{A}^0)}\big] \\
&= \mathbb{E}_q\big[\log \frac{p(\mathbf{A}^T)\prod_{t=1}^T p_\theta(\mathbf{A}^{t-1},\mathbf{s}^t|\mathbf{A}^t)}{\prod_{t=1}^T q(\mathbf{A}^t,\mathbf{s}^t|\mathbf{A}^{t-1})}\big] \\
&= \mathbb{E}_q\big[\log p(\mathbf{A}^T) + \sum_{t=1}^T \log \frac{p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t,\mathbf{s}^t)p_\theta(\mathbf{s}^t|\mathbf{A}^t)}{q(\mathbf{A}^t|\mathbf{A}^{t-1})q(\mathbf{s}^t|\mathbf{A}^t,\mathbf{A}^{t-1})}\big] \\
&= \mathbb{E}_q\big[\log p(\mathbf{A}^T) + \log \frac{p_\theta(\mathbf{A}^0|\mathbf{A}^1,\mathbf{s}^1)p_\theta(\mathbf{s}^1|\mathbf{A}^1)}{q(\mathbf{A}^1|\mathbf{A}^0)q(\mathbf{s}^1|\mathbf{A}^1,\mathbf{A}^0)} + \sum_{t=2}^T \log \frac{p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t,\mathbf{s}^t)p_\theta(\mathbf{s}^t|\mathbf{A}^t)}{\frac{q(\mathbf{A}^{t-1}|\mathbf{A}^t,\mathbf{s}^t,\mathbf{A}^0)q(\mathbf{A}^t|\mathbf{A}^0)q(\mathbf{s}^t|\mathbf{A}^t,\mathbf{A}^0)}{q(\mathbf{A}^{t-1}|\mathbf{A}^0)}}\big] \\
&= \mathbb{E}_q\big[\log p(\mathbf{A}^T) + \log \frac{p_\theta(\mathbf{A}^0|\mathbf{A}^1,\mathbf{s}^1)p_\theta(\mathbf{s}^1|\mathbf{A}^1)}{q(\mathbf{A}^1|\mathbf{A}^0)q(\mathbf{s}^1|\mathbf{A}^1,\mathbf{A}^0)} + \sum_{t=2}^T \log \frac{p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t,\mathbf{s}^t)p_\theta(\mathbf{s}^t|\mathbf{A}^t)q(\mathbf{A}^{t-1}|\mathbf{A}^0)}{q(\mathbf{A}^{t-1}|\mathbf{A}^t,\mathbf{s}^t,\mathbf{A}^0)q(\mathbf{s}^t|\mathbf{A}^t,\mathbf{A}^0)q(\mathbf{A}^t|\mathbf{A}^0)}\big] \\
&= \mathbb{E}_q\big[\log \frac{p(\mathbf{A}^T)\cancel{q(\mathbf{A}^T|\mathbf{A}^0)}}{q(\mathbf{A}^T|\mathbf{A}^0)} + \log \frac{p_\theta(\mathbf{A}^0|\mathbf{A}^1,\mathbf{s}^1)p_\theta(\mathbf{s}^1|\mathbf{A}^1)}{\cancel{q(\mathbf{A}^1|\mathbf{A}^0)}q(\mathbf{s}^1|\mathbf{A}^1,\mathbf{A}^0)} + \sum_{t=2}^T \log \frac{p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t,\mathbf{s}^t)p_\theta(\mathbf{s}^t|\mathbf{A}^t)\cancel{q(\mathbf{A}^{t-1}|\mathbf{A}^0)}}{q(\mathbf{A}^{t-1}|\mathbf{A}^t,\mathbf{s}^t,\mathbf{A}^0)q(\mathbf{s}^t|\mathbf{A}^t,\mathbf{A}^0)\cancel{q(\mathbf{A}^t|\mathbf{A}^0)}}\big] \\
&= \mathbb{E}_q\big[\log \frac{p(\mathbf{A}^T)}{q(\mathbf{A}^T|\mathbf{A}^0)} + \underbrace{\log p_\theta(\mathbf{A}^0|\mathbf{A}^1,\mathbf{s}^1)}_{\text{reconstruction term } \mathcal{L}_{\text{rec}}} + \sum_{t=2}^T \underbrace{\log \frac{p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t,\mathbf{s}^t)}{q(\mathbf{A}^{t-1}|\mathbf{A}^t,\mathbf{s}^t,\mathbf{A}^0)}}_{\text{edge prediction term } \mathcal{L}_{\text{edge}}(t)} + \sum_{t=1}^T \underbrace{\log \frac{p_\theta(\mathbf{s}^t|\mathbf{A}^t)}{q(\mathbf{s}^t|\mathbf{A}^t,\mathbf{A}^0)}}_{\text{node selection term } \mathcal{L}_{\text{node}}(t)}\big].
\end{aligned} \tag{30}
$$

The objective requires modeling two latent variables: $\mathbf{A}^{1:T}$ and $\mathbf{s}^{1:T}$. Learning to predict $\mathbf{s}^t$ from $\mathbf{A}^t$ can be difficult since it involves capturing the dynamic interaction between nodes and the global structure of the current graph $\mathbf{A}^t$. In Section B.2, we demonstrate a new objective which can avoid learning $p_\theta(\mathbf{s}^t|\mathbf{A}^t)$ by instead learning the node degree distribution $p_\theta(\mathbf{d}^0)$.

B.2. Derivation of the objective $\mathcal{L}(\mathbf{A}^0, \mathbf{d}^0; \theta)$

Since $p_\theta(\mathbf{A}^0) = p_\theta(\mathbf{A}^0, \mathbf{d}^0)$, we have

$$\begin{aligned}\log p_\theta(\mathbf{A}^0) = \log p_\theta(\mathbf{A}^0, \mathbf{d}^0) &\geq \mathcal{L}(\mathbf{A}^0, \mathbf{d}^0; \theta) \\ &= \mathbb{E}_q\big[\log \frac{p_\theta(\mathbf{d}^0)p_\theta(\mathbf{A}^0|\mathbf{d}^0)}{q(\mathbf{d}^0|\mathbf{A}^0)q(\mathbf{A}^{1:T}|\mathbf{A}^0)}\big] \\ &= \mathbb{E}_q\big[\log \frac{p_\theta(\mathbf{d}^0)p_\theta(\mathbf{A}^{0:T}, \mathbf{s}^{1:T}|\mathbf{d}^0)}{q(\mathbf{d}^0|\mathbf{A}^0)q(\mathbf{A}^{1:T}, \mathbf{s}^{1:T}|\mathbf{A}^0)}\big] \\ &= \underbrace{\mathbb{E}_q\big[\log \frac{p_\theta(\mathbf{d}^0)}{q(\mathbf{d}^0|\mathbf{A}^0)}\big]}_{\mathcal{L}(\mathbf{d}^0;\theta)} + \underbrace{\mathbb{E}_q\big[\log \frac{p_\theta(\mathbf{A}^{0:T}, \mathbf{s}^{1:T}|\mathbf{d}^0)}{q(\mathbf{A}^{1:T}, \mathbf{s}^{1:T}|\mathbf{A}^0)}\big]}_{\mathcal{L}(\mathbf{A}^0|\mathbf{d}^0;\theta)}.\end{aligned} \tag{31}$$

Optimizing $\mathcal{L}(\mathbf{d}^0; \theta)$ is equivalent to fitting $p_\theta(\mathbf{d}^0)$ to the node degree data distribution $p_{\mathrm{data}}(\mathbf{d}^0)$ as $\mathbf{d}^0$ is obtained from $\mathbf{A}^0$. The full decomposition of $\mathcal{L}(\mathbf{A}^0|\mathbf{d}^0; \theta)$ has the following form:

$$\mathcal{L}(\mathbf{A}^0|\mathbf{d}^0;\theta) = \mathbb{E}_q\big[\log \frac{p(\mathbf{A}^T)}{q(\mathbf{A}^T|\mathbf{A}^0)} + \underbrace{\log p_\theta(\mathbf{A}^0|\mathbf{A}^1, \mathbf{s}^1, \mathbf{d}^0)}_{\text{reconstruction term } \mathcal{L}_{\mathrm{rec}}} + \sum_{t=2}^{T} \underbrace{\log \frac{p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{s}^t, \mathbf{d}^0)}{q(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{s}^t, \mathbf{A}^0)}}_{\text{edge prediction term } \mathcal{L}_{\text{edge}}(t)} + \sum_{t=1}^{T} \underbrace{\log \frac{p_\theta(\mathbf{s}^t|\mathbf{A}^t, \mathbf{d}^0)}{q(\mathbf{s}^t|\mathbf{A}^t, \mathbf{A}^0)}}_{\text{node selection term } \mathcal{L}_{\text{node}}(t)}\big]. \tag{32}$$

Here $\mathbf{A}^T$ is independent from $\mathbf{d}^0$ so $p(\mathbf{A}^T|\mathbf{d}^0) = p(\mathbf{A}^T)$. And as mentioned before, we choose to parameterize $p_\theta(\mathbf{s}^t|\mathbf{A}^t, \mathbf{d}^0) := q(\mathbf{s}^t|\mathbf{A}^t, \mathbf{A}^0)$, resulting in the KL divergence $\mathcal{L}_{\text{node}}(t) = 0$ for all t. The objective is further simplified to

$$\mathcal{L}(\mathbf{A}^0|\mathbf{d}^0; \theta) = \mathbb{E}_q\big[\log \frac{p(\mathbf{A}^T)}{q(\mathbf{A}^T|\mathbf{A}^0)} + \log p_\theta(\mathbf{A}^0|\mathbf{A}^1, \mathbf{s}^1, \mathbf{d}^0) + \sum_{t=2}^{T} \log \frac{p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{s}^t, \mathbf{d}^0)}{q(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{s}^t, \mathbf{A}^0)}\big]. \tag{33}$$

C. Detailed Implementations of the Denoising Networks and Training

C.1. Parameterization of the edge prediction distribution

As we consider modeling the upper triangle of the adjacency matrix, the edge prediction distribution $p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{s}^t, \mathbf{d}^0)$ is parameterized as:

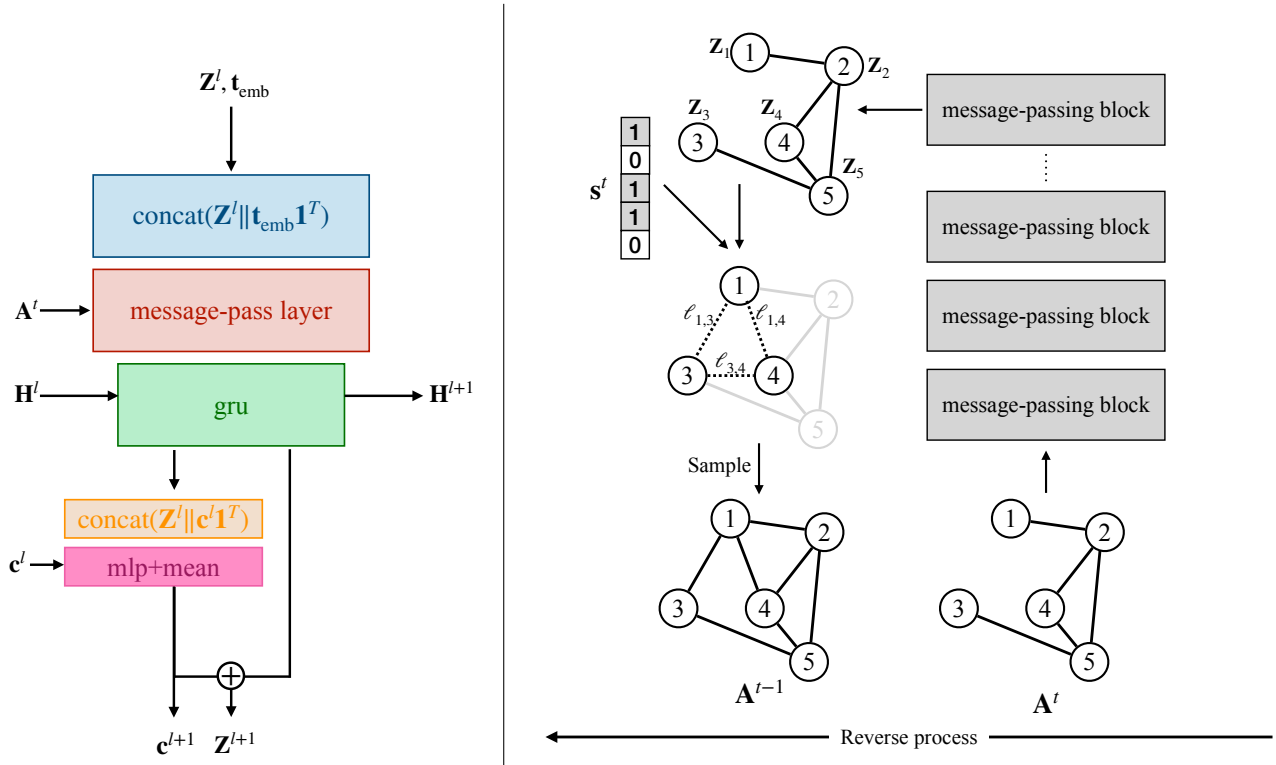
$$\begin{aligned}&p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{s}^t, \mathbf{d}^0) = \prod_{i,j:i<j} p_\theta(\mathbf{A}_{i,j}^{t-1}|\mathbf{A}_{i,j}^t, \mathbf{s}_{i,j}^t, \mathbf{d}^0), \text{ with} \\ &p_\theta(\mathbf{A}_{i,j}^{t-1}|\mathbf{A}_{i,j}^t, \mathbf{s}_{i,j}^t, \mathbf{d}^0) = \mathcal{B}(\mathbf{A}_{i,j}^t; \mathbf{s}_{i,j}^t \ell_{i,j}^{t-1} + (1 - \mathbf{s}_{i,j}^t)\mathbf{A}_{i,j}^t), \\ &\text{where } \mathbf{s}_{i,j}^t = \mathbf{s}_i^t \mathbf{s}_j^t,\ \ell^{t-1} = \mathrm{gnn}_\theta(\mathbf{A}^t, \mathbf{s}^t, \mathbf{d}^0, t).\end{aligned} \tag{34}$$

Note that only when nodes i, j are both selected, i.e., $\mathbf{s}_{i,j}^t = 1$, the corresponding Bernoulli parameter $\ell_{i,j}^{t-1}$ effectively decides the edge distribution. Below we elaborate on the architecture of the used network and analyze the runtime complexity.

Architecture design and complexity analysis. Figure 5 shows the parameterization and the inference procedure of the edge prediction model. The main component of the parameterized network consists of L message-passing blocks (MPB). After constructing the inputs, we iteratively update the node features using MPBs, and then compute the Bernoulli parameter $\ell_{i,j}^{t-1}$ for node pairs (i, j) using a Multilayer perceptron(MLP). The computation flow is shown below

$$\begin{aligned}&\mathbf{Z}^0 = \mathrm{concat}(\mathrm{emb}(\mathbf{d}^t)\|\mathrm{emb}(\mathbf{d}^0)),\ \mathbf{t}_{\mathrm{emb}} = \mathrm{emb}(t),\ \mathbf{c}^0 = \mathrm{mean}(\mathbf{Z}^0); \\ &\mathbf{Z}^l, \mathbf{c}^l, \mathbf{H}^l = \mathrm{block}_l(\mathbf{Z}^{l-1}, \mathbf{t}_{\mathrm{emb}},\ \mathbf{c}^{l-1}, \mathbf{H}^{l-1}),\ \text{for } l = 1, \ldots, L; \\ &\ell_{i,j}^{t-1} = \mathrm{mlp}(\mathbf{Z}_i^L + \mathbf{Z}_j^L),\ \text{where } (i,j) \in \{(i,j)|\mathbf{s}_{i,j}^t = 1, 1 \leq i < j \leq N\}.\end{aligned}$$

Here $\mathbf{Z}^0$ is the node features initialized using degree sequence $\mathbf{d}^0$ and $\mathbf{d}^t$; $\mathbf{c}$ is the global context features; $\mathbf{H}$ is the node hidden state features. We follow Dhariwal & Nichol (2021) and use the sinusoidal position embedding for the diffusion



(a) Message-passing blocks

(b) Edge prediction of one step

Figure 5. Parameterization and inference of the edge prediction network. (a) shows the architecture of an MPB, which updates the node features $\mathbf{Z}$ and context feature $\mathbf{c}$ via message-passing. The state vectors $\mathbf{H}$ are used for keeping geometric information at different levels of layers. (b) shows how the model infers edges during the reverse process. The node features are computed using a stacked MPB. The model only predicts edges between active nodes indicated by $\mathbf{s}^t$.

timestep t. $\mathbf{Z}$, $\mathbf{c}$, and $\mathbf{H}$ are iteratively updated by MPBs. Finally, the edge prediction parameter $\ell_{i,j}^{t-1}$ is computed from the two node embeddings.

Each MPB contains a message-passing layer (MPL) and a Gated Recurrent Unit (GRU) (Cho et al., 2014). We experimented on various MPLs and found Unified Message Passaging Model (Shi et al., 2020) demonstrates better expressivity for our tasks. The details for the l-th MBP can be represented as follow

$$
\begin{aligned}
\mathbf{Z}^l &= \text{concat}(\mathbf{Z}^l \| \mathbf{t}_{\text{emb}} \mathbf{1}^T) \\
\mathbf{Z}^l &= \text{mpl}(\mathbf{Z}^l, \mathbf{A}^t) \\
\mathbf{Z}^l, \mathbf{H}^{l+1} &= \text{gru}(\mathbf{Z}^l, \mathbf{H}^l) \\
\mathbf{c}^{l+1} &= \text{mlp}(\text{mean}(\text{concat}(\mathbf{Z}^l \| \mathbf{c}^l \mathbf{1}^T))) \\
\mathbf{Z}^{l+1} &= \mathbf{Z}^l + \mathbf{c}^{l+1} \mathbf{1}^T
\end{aligned}
$$

Note that in MPB, the message-passing operation has runtime complexity $O(M)$, and all other operations have runtime complexity $O(N)$ as they operate on node features. The total runtime complexity of an MPB module is $O(M)$. Another major computation comes from the edge probability predictions, which are decided by the size of the node-pair set $|\{(i,j) | \mathbf{s}_{i,j}^t = 1, i < j\}| = (\sum_i \mathbf{s}_i^t)^2 \leq K^2$.

C.2. Modeling the degree sequence distribution

We want the generative model $p_\theta(\mathbf{d}^0)$ is invariant on node permutation π, i.e., for all π, we have $p_\theta(\mathbf{d}^0) = p_\theta(\pi(\mathbf{d}^0))$. Let $\mathbf{u} \in \{0, \ldots, N_{\max}\}^{d_{\max}}$ be a histogram such that $\mathbf{u}_k = \sum_{i=1}^N \mathbf{1}[\mathbf{d}_i^0 = k]$ for $k = 1, \ldots, d_{\max}$, and $d_{\max}$ is the maximum

node degree in the graph training set. We consider modeling $q(\mathbf{u}|\mathbf{A}^0)$ instead of $q(\mathbf{d}^0|\mathbf{A}^0)$ as $\mathbf{u}$ is invariant to node permutation. The training objective is then to fit $p_\theta(\mathbf{u})$ to the empirical data distribution $p_{\text{data}}(\mathbf{u})$. We consider it a sequence modeling task. The model $p_\theta(\mathbf{u})$ has the following decomposition

$$p_\theta(\mathbf{u}) = p_\theta(\mathbf{u}_1) \prod_{k=2}^{d_{\max}} p_\theta(\mathbf{u}_k|\mathbf{u}_{<k}). \tag{35}$$

We parameterize $p_\theta(\mathbf{u}_k|\mathbf{u}_{<k})$ with a RNN (Schuster & Paliwal, 1997), where the RNN produces $N_{\max}$-dimensional softmax logits at each step. Once a sequence $\mathbf{u}$ is sampled, the graph's corresponding size is known. Note that we only model the degree distribution for the generic graph generation task. We only have one graph for large network generation for each dataset. We use the node degrees of the training graph when sampling new graphs. We describe the sampling algorithm in Alg. 2.

Algorithm 2 Sampling the degree sequence $\mathbf{d}^0$

Input: Maximum degree $d_{\max}$, degree sequence model $p_\theta(\mathbf{u})$.
Draw $\mathbf{u}_1 \sim p_\theta(\mathbf{u}_1)$, initialize $\mathbf{d}^0 = \mathbf{1}_{\mathbf{u}_1}$.
for $k = 2 \ldots, d_{\max}$ **do**
  Initialize $\mathbf{u}_k$-dimensional all-ones vector $\mathbf{1}_{\mathbf{u}_k}$, set $\mathbf{d}^0 = \text{concat}(\mathbf{d}^0 \| k\mathbf{1}_{\mathbf{u}_k})$.
  Draw $\mathbf{u}_k \sim p_\theta(\mathbf{u}_k|\mathbf{u}_{<k})$
end for
Output: Degree sequence $\mathbf{d}^0$

C.3. Training

We demonstrate the training of the edge prediction model $p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{s}^t, \mathbf{d}^0)$. Since we do not have a closed-form computation of the expectation, the gradient $\nabla_\theta \mathcal{L}(\mathbf{A}^0; \theta, \mathbf{d}^0)$ is estimated via Monte Carlo samples:

$$\nabla_\theta \mathcal{L}(\mathbf{A}^0; \theta, \mathbf{d}^0) = \mathbb{E}_{t\sim\mathcal{U}[1,T]}\Big[\mathbb{E}_{q(\mathbf{A}^{t-1}|\mathbf{A}^0)}\mathbb{E}_{q(\mathbf{A}^t|\mathbf{A}^{t-1})}\mathbb{E}_{q(\mathbf{s}^t|\mathbf{A}^{t-1},\mathbf{A}^t)}\big[\log p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{s}^t, \mathbf{d}^0)\big]\Big]. \tag{36}$$

We describe the procedure of training the edge prediction model in Alg. 3

Algorithm 3 Training procedure for edge prediction model

Input: Data distribution $p_{\text{data}}(\mathbf{A}^0)$, diffusion steps T, forward transition distributions $q(\mathbf{A}^t|\mathbf{A}^{t-1})$ and $q(\mathbf{A}^t|\mathbf{A}^0)$.
while training **do**
  Draw $\mathbf{A}^0 \sim p_{\text{data}}(\mathbf{A}^0)$, compute node degree $\mathbf{d}^0 = \deg(\mathbf{A}^0)$.
  Sample $t \sim \mathcal{U}[1, T]$, and have $t-1$ immediately.
  Draw $\mathbf{A}^{t-1} \sim q(\mathbf{A}^{t-1}|\mathbf{A}^0)$, then draw $\mathbf{A}^t \sim q(\mathbf{A}^t|\mathbf{A}^{t-1})$.
  Obtain active node variable $\mathbf{s}^t$ from $\mathbf{A}^{t-1}, \mathbf{A}^t$.
  Perform gradient descent over θ with gradient $\nabla_\theta \log p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{s}^t, \mathbf{d}^0)$
end while

D. Further Discussion of Current Diffusion Graph Models

D.1. Technical differences of existing diffusion-based graph models

Diffusion-based generative models (Sohl-Dickstein et al., 2015; Ho et al., 2020; Hoogeboom et al., 2021; Austin et al., 2021) have gained prominent attention in the graph generation community. Current diffusion graph models adopt continuous or discrete-variable diffusion. For example, GDSS (Jo et al., 2022) and EDP-GNN (Niu et al., 2020) use Gaussian transition kernels for continuous-variable diffusion (Ho et al., 2020), but GDSS additionally considers node and edge features and uses a score-matching diffusion approach. On the other hand, DiscDDPM (Haefeli et al., 2022) and DiGress (Vignac et al., 2022) employ discrete-time, discrete-variable diffusion models (Hoogeboom et al., 2021; Austin et al., 2021), with the former being featureless and the latter including a denoising process for node and edge attributes.

One limitation shared by all current diffusion graph models is suffering from the scalability issue: they all need $O(TN^2)$ running time because they need to make predictions for each node pair in each diffusion step. This limitation prevents diffusion graph models from generating large networks. The main advantage of EDGE is to reduce this complexity to $O(\min(K^2, M))$, with K being the number of active nodes and M being the number of edges. It greatly reduces the computation in the generation process and has shown promising results in large network generation tasks. We further highlight the limitation and technical differences of different models in Table 7.

	Diffusion type	Convergent distribution	Conditional generation	Featured graph generation	Runtime	Scalability
EDP-GNN	Disc. time, Cont. var.	$\mathcal{N}(0,1)$			$O(TN^2)$	
GDSS	Cont. time, Cont. var.	$\mathcal{N}(0,1)$		✓	$O(TN^2)$	
DiscDDPM	Disc. time, Disc. var.	$G(N, 0.5)$			$O(TN^2)$	
DiGress	Disc. time, Disc. var.	Empirical distribution	Gradient from a classifier	✓	$O(TN^2)$	
EDGE (ours)	Disc. time, Disc. var.	$G(N, 0)$	degree sequence		$O(T\max(M, K^2))$	✓

Table 7. Technical differences of different diffusion graph models. Here T is the number of diffusion steps, N is the number of nodes in a graph, M is the number of edges in a graph, and K is the maximum number of active nodes during the diffusion process.

E. Generalizing to Tasks of Generating Attributed Graphs

E.1. Hierarchical generation

Generation of attributed graphs has a broad class of applications, such as molecule generation (Du et al., 2021). While EDGE is developed to generate graph structure only, here we briefly discuss how it can be incorporated into a hierarchical procedure to generate graphs with node and edge attributes. Here we consider the case where node and edge attributes are both categorical.

The attributes of nodes and edges are represented as one-hot vectors. For node attributes, we have a matrix $\mathbf{X} \in \{0,1\}^{N \times \mathcal{C}_{\text{node}}}$, while edge attributes are described by the matrix $\mathbf{A}_{\text{attr}} \in \{0,1\}^{N \times (\mathcal{C}_{\text{edge}}+1)}$. In this context, $\mathcal{C}_{\text{node}}$ and $\mathcal{C}_{\text{edge}}$ denote the number of classes for node types and edge types, respectively. For a node pair (i, j), the extra dimension indicates whether the edge exists or not. The graph structure is still denoted by $\mathbf{A}$. Inspired by Lippe & Gavves (2020), we consider the following joint model:

$$p(\mathbf{X}, \mathbf{A}_{\text{attr}}, \mathbf{A}) = p(\mathbf{X})p(\mathbf{A}|\mathbf{X})p(\mathbf{A}_{\text{attr}}|\mathbf{X}, \mathbf{A}), \tag{37}$$

which can be considered a hierarchical generation scheme that first samples node attributes, then samples the graph structure via EDGE conditioned on node attributes, and finally samples edge attributes conditioned on the graph and node attributes.

E.2. Model Details

We consider modeling each component in Eqn. (37) separately. For $p(\mathbf{X})$, we employ a similar approach as with the node degree sequence modeling, but we use the sequence length $\mathcal{C}_{\text{node}}$ instead of $d_{\max}$. For $p(\mathbf{A}|\mathbf{X})$, we apply the EDGE framework, incorporating node features from $\mathbf{X}$ during both the training and generation phases. For $p(\mathbf{A}_{\text{attr}}|\mathbf{X}, \mathbf{A})$, we utilize a diffusion model that starts by randomly assigning edge types to edges in $\mathbf{A}$ and iteratively refines edge labels, relying on the information given by $\mathbf{X}$ and $\mathbf{A}$. It is important to note that we only refine labels for edges already specified by $\mathbf{A}$, allowing us to use an MPNN to calculate edge features. We adopt the framework outlined in Appendix C, and only perform prediction for edges existing in $\mathbf{A}$.

F. Additional Details for Experimental Setups

We described the details of the experiments of generic graph generation and large network generation tasks. We provide the hyperparameters used in the experiments in Table 8. We do not augment the data input with extra features for all generation

tasks except for the current node degrees $\mathbf{d}^t$ and the node degrees $\mathbf{d}^0$, which are both computation-free. Moreover, we set $p = 10^{-12}$ in our implementation to maintain numerical stability.

	Community	Ego	Polblogs	Cora	Road-Minnesota	PPI
Diffusion						
Diffusion steps T	128	128	256	64	64	512
Noise scheduling	Linear					
β_0	7.8125×10^{-4}		3.9063×10^{-4}	1.5625×10^{-3}		1.9531×10^{-4}
β_T	1.5625×10^{-1}		7.8125×10^{-2}	3.1250×10^{-1}		3.9063×10^{-2}
Sample time method	Importance sampling					
Optimization						
Learning rate	10^{-4}					
Optimizer	Adam (Kingma & Ba, 2014)					
weight decay	10^{-4}					
Batch size	64	64	4	4	4	1
Number of epochs/iteration	30000	10000	50000	50000	50000	50000
Architecture						
Number of MPBs	5					
Hidden dimension of MPL	64					
Hidden dimension of GRU	64					
Activation function	SiLU (Elfwing et al., 2018)					
Time embedding	Sinusoidal positional embedding (Devlin et al., 2018)					
Dropout rate	0.1					
Evaluation						
Number of generated graphs	128	128	5	5	5	5
$d_{\max}$	40	100	351	168	5	593
Number of attention heads	8	8	8	8	8	8

Table 8. Hyperparameters

F.1. Generic graph generation

We follow You et al. (2018) to generate the Community and Ego datasets and use the same data splitting strategy. Recent works (O'Bray et al., 2021; Thompson et al., 2022) have suggested better metrics for evaluating the quality of the generated graphs. To make a fair comparison, we reproduce all baselines and follow Thompson et al. (2022) to re-evaluate their generative performance. All the baselines are reproduced using their default hyperparameter setting except for GraphCNF and DiGress. For GraphCNF, we use the same model configuration of its molecule generation task for the Community dataset and a smaller model for the Ego dataset due to the limited capacity of the GPU memory. For DiGress, we do not augment the graphs with the structural features to ensure a fair comparison is made.

F.2. Large network generation

We consider the single network for each large network dataset as the training dataset. Since the evaluation metrics do not require referring to the test graphs, we do not include validation/test sets in this task. All models are trained until the network statistics converge, and the models of the final epoch are used to generate samples. For GraphRNN, we use the default BFS ordering to generate adjacency matrices for the model training. We train the model for 30000 iterations for all datasets and report the model performance using the checkpoint from the last epoch.

Computing the Edge Overlap. Since GraphRNN and our model are edge non-independent models, Chanpuriya et al. (2021) suggests reporting the maximum edge overlap between the generated graphs and the training graph to ensure the models do not simply memorize the data. However, finding the maximum edge overlap requires searching over the node permutation space, which is impractical as there are $N!$ permutations. Instead, we obtain the node degree ascending permutation and use it to permute both the generated and training graphs. We observe that such a permutation scheme yields

a much higher EO value than a random permutation. For instance, when a model can generate graphs with desiring statistics, degree-based permutation yields 15% EO on average for the Poblogs dataset, while a random permutation yields an EO value that is almost 0.

F.3. Computational Resources

We use PyTorch (Paszke et al., 2019) and PyTorch Geometric (Fey & Lenssen, 2019) to implement our framework. We train our models on Tesla A100, Tesla V100, or NVIDIA QUADRO RTX 6000 GPU and 32 CPU cores for all experiments. For generic graph generation tasks, all models are trained within 72 hours. For large network generation tasks, model training is finished within 24 hours. The sampling speed reported in Figure 3 of all baselines and our approach is tested on Tesla A100 GPU.

G. Extended Results

G.1. Comparison between the node degrees from generated graphs and the node degrees $\mathbf{d}^0$

We show that the generated graphs' node degrees accurately approximate the given node degrees $\mathbf{d}^0$. The node degrees in the generated graphs are compared to the node degrees $\mathbf{d}^0$ by counting the number of nodes whose degree deviates from the given one. The degree difference is computed by subtracting the given degree from the actual degree. The histograms in Figure 6 display the degree difference for each dataset, indicating the accuracy of the generated graph's node degrees in approximating the given node degrees.

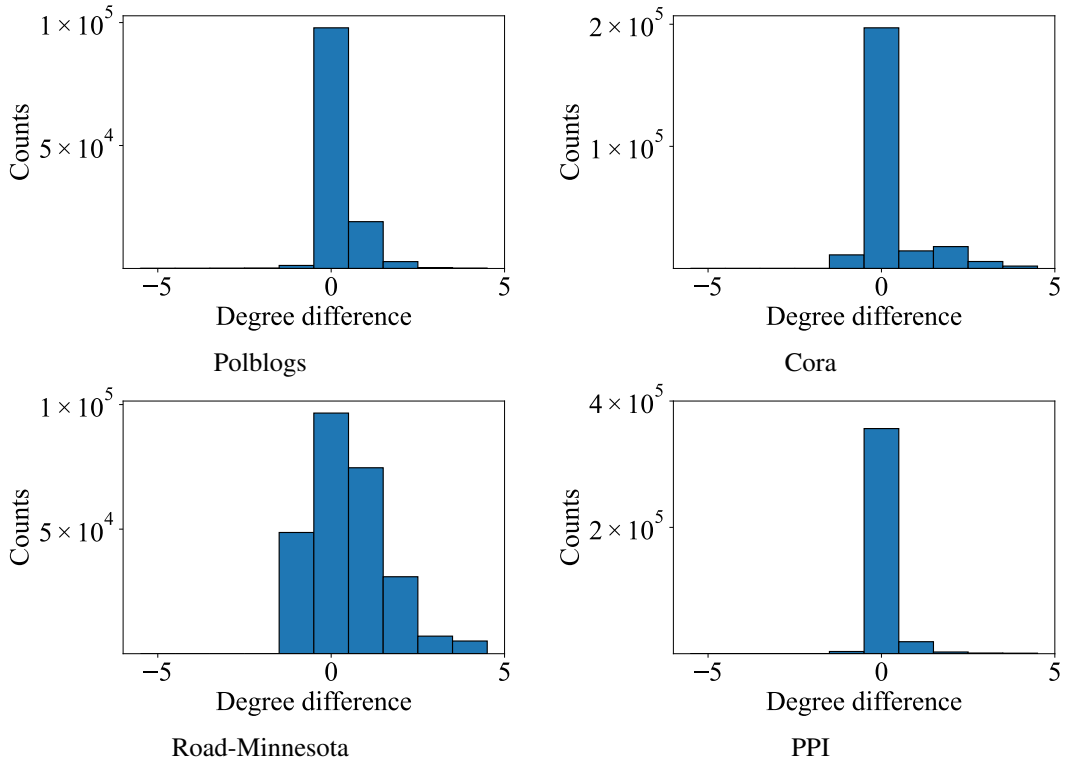


Figure 6. Degree difference. Given specific node degrees $\mathbf{d}^0$, the actual node degrees of the generated graphs are fairly accurate.

G.2. Further justification the use of $G(N, 0)$ as the convergent distribution

In addition to the desirable properties we described in Section 3.1, we demonstrate the potential benefit of using $G(N, 0)$ as the convergent distribution in terms of generative performance. When using $G(N, 0)$ as the convergent distribution, our proposed framework can be considered as a type of absorbing diffusion process (Austin et al., 2021). Similar to (Austin et al., 2021), we observe that the generative performance of $G(N, 0)$ is superior to $G(N, 0.5)$. Table 9 report the

generative performance of $G(N, 0.5)$ and $G(N, 0)$ on the Community-small dataset (You et al., 2018). This demonstrates the superiority of using an absorbing state as the convergent distribution, which further justifies why one should consider $G(N, 0)$ as the convergent distribution.

	Community-small				
	Structure-based metrics (MMD)			Neural-based metrics	
	Deg.	Clus.	Orb.	FID	RBF MMD
$G(N, 0.5)$	0.0274	0.0249	**0.0234**	3.4121	0.0243
$G(N, 0)$	**0.0081**	**0.0112**	0.0262	**1.5642**	**0.0204**

Table 9. Vanilla discrete diffusion with $G(N, 0.5)$ and $G(N, 0)$ as the convergent distributions. $G(N, 0)$ exhibits better generative performance than $G(N, 0.5)$.

G.3. Full results on graph generation tasks

We provide the mean and the standard derivation of metrics reported in the generic graph generation and large network generation tasks in Table 10 and Table 11, respectively.

	Community				
	Structure-based metrics (MMD)			Neural-based metrics	
	Deg.	Clus.	Orb.	FID	RBF MMD
GRNN	0.1440 ± 0.0025	0.0535 ± 0.0264	$\mathbf{0.0198} \pm \mathbf{0.0003}$	8.3869 ± 1.5429	0.1591 ± 0.0104
GRAN	0.1022 ± 0.0185	0.0894 ± 0.0082	$\mathbf{0.0198} \pm \mathbf{0.0005}$	64.1145 ± 12.0927	0.0749 ± 0.0097
GraphCNF	0.1129 ± 0.0295	1.2882 ± 0.1918	$\mathbf{0.0197} \pm \mathbf{0.0005}$	29.1526 ± 3.1900	0.1341 ± 0.0241
GDSS	0.0535 ± 0.0095	0.2072 ± 0.0520	$\mathbf{0.0196} \pm \mathbf{0.0003}$	6.5531 ± 0.9418	0.0443 ± 0.0058
DiscDDPM	0.1238 ± 0.0068	0.6549 ± 0.0463	0.0246 ± 0.0004	8.6321 ± 1.1961	0.0840 ± 0.0099
DiGress	0.0409 ± 0.0041	$\mathbf{0.0167} \pm \mathbf{0.0169}$	0.0298 ± 0.0002	3.4261 ± 0.4549	0.0460 ± 0.0069
EDGE	$\mathbf{0.0175} \pm \mathbf{0.0056}$	0.0689 ± 0.0197	$\mathbf{0.0198} \pm \mathbf{0.0002}$	$\mathbf{2.2378} \pm \mathbf{0.5111}$	$\mathbf{0.0227} \pm \mathbf{0.0097}$

	Ego				
	Structure-based metrics (MMD)			Neural-based metrics	
	Deg.	Clus.	Orb.	FID	RBF MMD
GRNN	0.0768 ± 0.0142	1.1456 ± 0.0910	0.1087 ± 0.0442	90.5655 ± 19.2041	0.6827 ± 0.1181
GRAN	0.5778 ± 0.1415	0.3360 ± 0.0948	$\mathbf{0.0406} \pm \mathbf{0.0112}$	489.9598 ± 42.1109	0.2633 ± 0.0911
GraphCNF	0.1010 ± 0.0421	0.7654 ± 0.0510	0.0820 ± 0.0334	18.7929 ± 3.5102	0.0896 ± 0.0125
GDSS	0.8189 ± 0.0691	0.6032 ± 0.2114	0.3315 ± 0.0591	60.6100 ± 8.1208	0.4331 ± 0.0982
DiscDDPM	0.4613 ± 0.1042	0.1681 ± 0.0735	0.0633 ± 0.0156	42.7994 ± 5.6312	0.1561 ± 0.0224
DiGress	0.0708 ± 0.0127	$\mathbf{0.0092} \pm \mathbf{0.0062}$	0.1205 ± 0.0669	18.6794 ± 4.6395	$\mathbf{0.0489} \pm \mathbf{0.0232}$
EDGE	$\mathbf{0.0579} \pm \mathbf{0.0101}$	0.1773 ± 0.0521	$\mathbf{0.0519} \pm \mathbf{0.0216}$	$\mathbf{15.7614} \pm \mathbf{2.5021}$	$\mathbf{0.0658} \pm \mathbf{0.0199}$

Table 10. Generation performance on generic graphs with standard derivation.

G.4. Visualizations

Visualization of generated generic graphs. We visualize six generic graphs from the test data and the generated graphs for each dataset in Figure 7 and 8. The visualized graphs are randomly selected from the test data and the generated samples.

Visualization of generated molecules. We visualize 16 molecules generated from GDSS, DiGress, and our approach in Figure 9.

Polblogs	EO	PLE	NTC	CC	CPL	AC
TRUE	100	1.414	1	0.226	2.736	-0.221
OPB	24.5 ± 0.4	1.395 ± 0.002	0.667 ± 0.013	0.150 ± 0.001	2.524 ± 0.005	-0.143 ± 0.003
HDOP	16.4 ± 0.3	1.393 ± 0.003	0.687 ± 0.021	0.153 ± 0.002	2.522 ± 0.009	-0.131 ± 0.006
CELL	26.8 ± 0.2	1.385 ± 0.001	0.810 ± 0.011	0.211 ± 0.002	2.536 ± 0.006	-0.230 ± 0.002
CO	20.1 ± 0.2	1.975 ± 0.107	0.045 ± 0.002	0.028 ± 0.001	2.502 ± 0.008	0.068 ± 0.009
TSVD	32.0 ± 0.2	1.373 ± 0.001	0.872 ± 0.023	0.205 ± 0.004	2.533 ± 0.005	**-0.216 ± 0.005**
VGAE	3.6 ± 0.2	1.723 ± 0.010	0.05 ± 0.006	0.001 ± 0.001	2.531 ± 0.063	-0.086 ± 0.009
GRNN	9.6 ± 0.5	1.334 ± 0.013	0.355 ± 0.048	0.095 ± 0.008	2.566 ± 0.056	0.096 ± 0.065
EDGE	16.5 ± 0.3	**1.398 ± 0.002**	**0.977 ±0.079**	**0.217 ± 0.005**	**2.647 ± 0.028**	**-0.214 ± 0.015**

Cora	EO	PLE	NTC	CC	CPL	AC
TRUE	100	1.885	1	0.090	6.311	-0.071
OPB	10.9 ± 0.2	**1.852 ± 0.008**	0.097 ± 0.019	0.008 ± 0.001	4.476 ± 0.046	-0.037 ± 0.009
HDOP	0.9 ± 0.1	**1.849 ± 0.011**	0.113 ± 0.003	0.009 ± 0.001	4.477 ± 0.030	-0.030 ± 0.004
CELL	10.3 ± 0.2	1.774 ± 0.001	0.009 ± 0.003	0.002 ± 0.001	5.799 ± 0.012	-0.018 ± 0.013
CO	9.7 ± 0.5	1.776 ± 0.007	0.009 ± 0.002	0.002 ± 0.000	5.653 ± 0.044	0.010 ± 0.012
TSVD	6.7 ± 0.2	**1.858 ± 0.012**	0.349 ± 0.029	0.028 ± 0.001	4.908 ± 0.052	-0.006 ± 0.005
VGAE	1.5 ± 0.5	1.717 ± 0.005	0.120 ± 0.012	0.220 ± 0.012	4.934 ± 0.069	0.002 ± 0.010
GRNN	0.4 ± 0.1	1.822 ± 0.008	0.043 ± 0.007	0.011 ± 0.002	**6.146 ± 0.065**	0.043 ± 0.025
EDGE	0.9 ± 0.0	1.755 ± 0.005	**0.446 ± 0.029**	**0.034 ± 0.002**	4.995 ± 0.048	**-0.046 ± 0.008**

Road-Minnesota	EO	PLE	NTC	CC	CPL	AC
TRUE	100	2.147	1	0.028	35.349	-0.187
OPB	29.7 ± 0.3	**2.188 ± 0.016**	0.083 ± 0.036	0.002 ± 0.001	8.036 ± 0.051	0.009 ± 0.011
HDOP	13.2 ± 1.1	**2.192 ± 0.065**	0.208 ± 0.111	0.004 ± 0.001	8.274 ± 0.032	-0.024 ± 0.006
CELL	30.7 ± 1.3	2.267 ± 0.011	0.053 ± 0.069	0.001 ± 0.001	10.219 ± 0.096	**-0.082 ± 0.00**4
CO	19.8 ± 0.9	2.044 ± 0.049	2.845 ± 0.916	**0.040 ± 0.003**	11.478 ± 0.075	-0.012 ± 0.008
TSVD	19.4 ± 0.6	**2.172 ± 0.041**	0.060 ± 0.046	0.001 ± 0.000	8.431 ± 0.130	0.006 ± 0.009
VGAE	1.3 ± 0.3	1.678 ± 0.091	0.096 ± 0.031	0.009 ± 0.001	11.120 ± 0.075	-0.027 ± 0.001
GRNN	0.6 ± 0.1	1.570 ± 0.017	0.099 ± 0.023	0.007 ± 0.002	**11.695 ± 0.059**	0.006 ± 0.009
EDGE	0.8 ± 0.1	1.910 ± 0.023	**0.962 ± 0.101**	0.011 ± 0.001	9.125 ± 0.088	-0.063 ± 0.006

PPI	EO	PLE	NTC	CC	CPL	AC
TRUE	100	1.462	1	0.092	3.095	-0.099
OPB	16.3 ± 0.2	1.443 ± 0.001	0.640 ± 0.007	0.058 ± 0.000	2.914 ± 0.005	**-0.089 ± 0.003**
HDOP	6.9 ± 0.1	1.444 ± 0.001	0.638 ± 0.007	0.058 ± 0.001	2.917 ± 0.008	-0.086 ± 0.003
CELL	6.7 ± 0.2	1.400 ± 0.000	0.248 ± 0.005	0.040 ± 0.001	**3.108 ± 0.003**	0.176 ± 0.004
CO	9.9 ± 0.1	1.754 ± 0.071	0.016 ± 0.001	0.006 ± 0.000	3.046 ± 0.002	0.043 ± 0.004
TSVD	13.2 ± 0.1	1.426 ± 0.001	0.848 ± 0.015	0.077 ± 0.001	2.867 ± 0.004	**-0.089 ± 0.004**
VGAE	0.5 ± 0.0	1.362 ± 0.006	0.091 ± 0.009	0.012 ± 0.005	2.991 ± 0.063	0.054 ± 0.007
GRNN	OOM	OOM	OOM	OOM	OOM	OOM
EDGE	7.5 ± 0.4	**1.449 ± 0.003**	**0.981 ± 0.003**	**0.091 ± 0.031**	3.028 ± 0.044	**-0.107 ± 0.023**

Table 11. Generation performance on large networks with standard derivation.

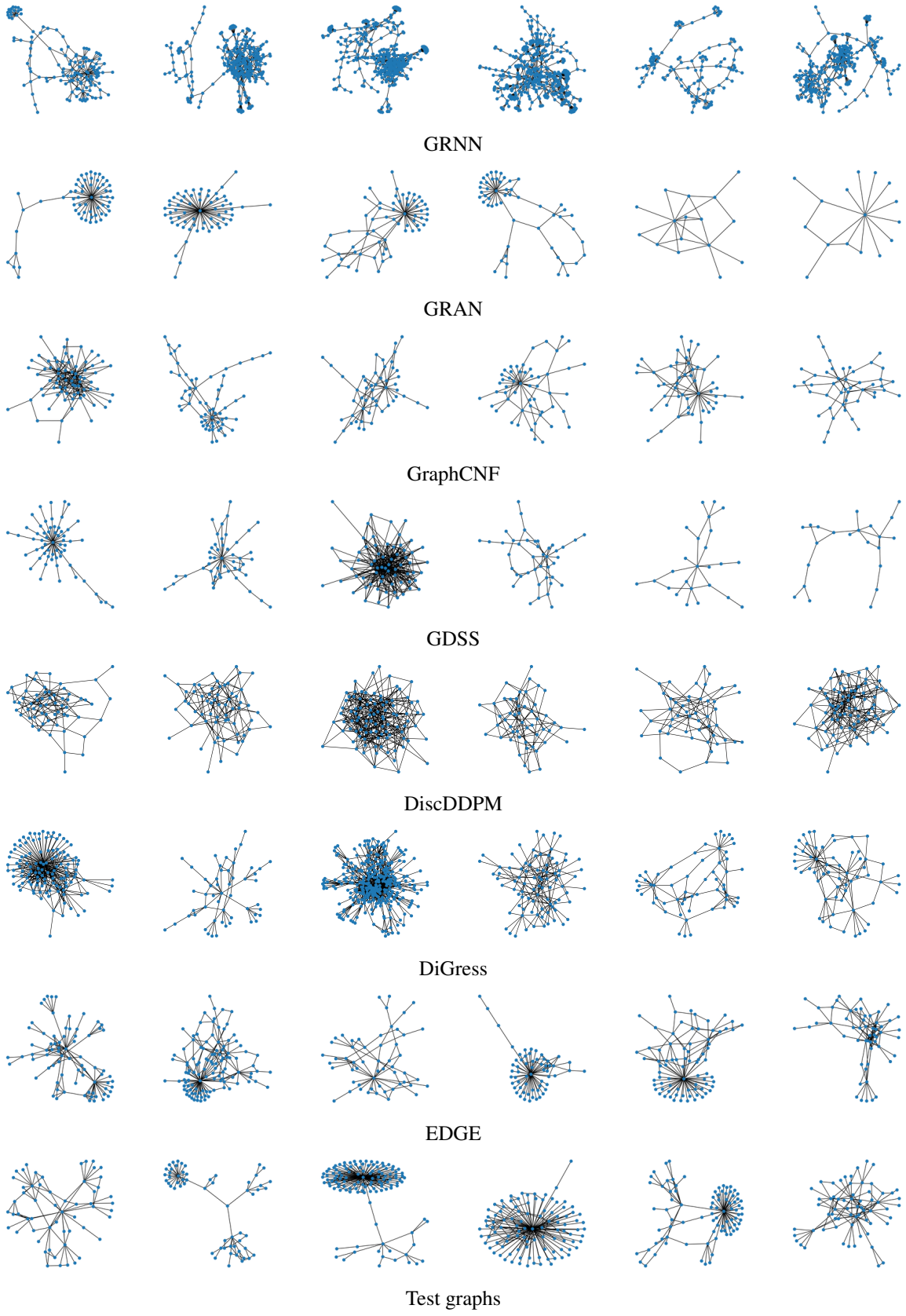


Figure 7. Visualization of the Ego dataset

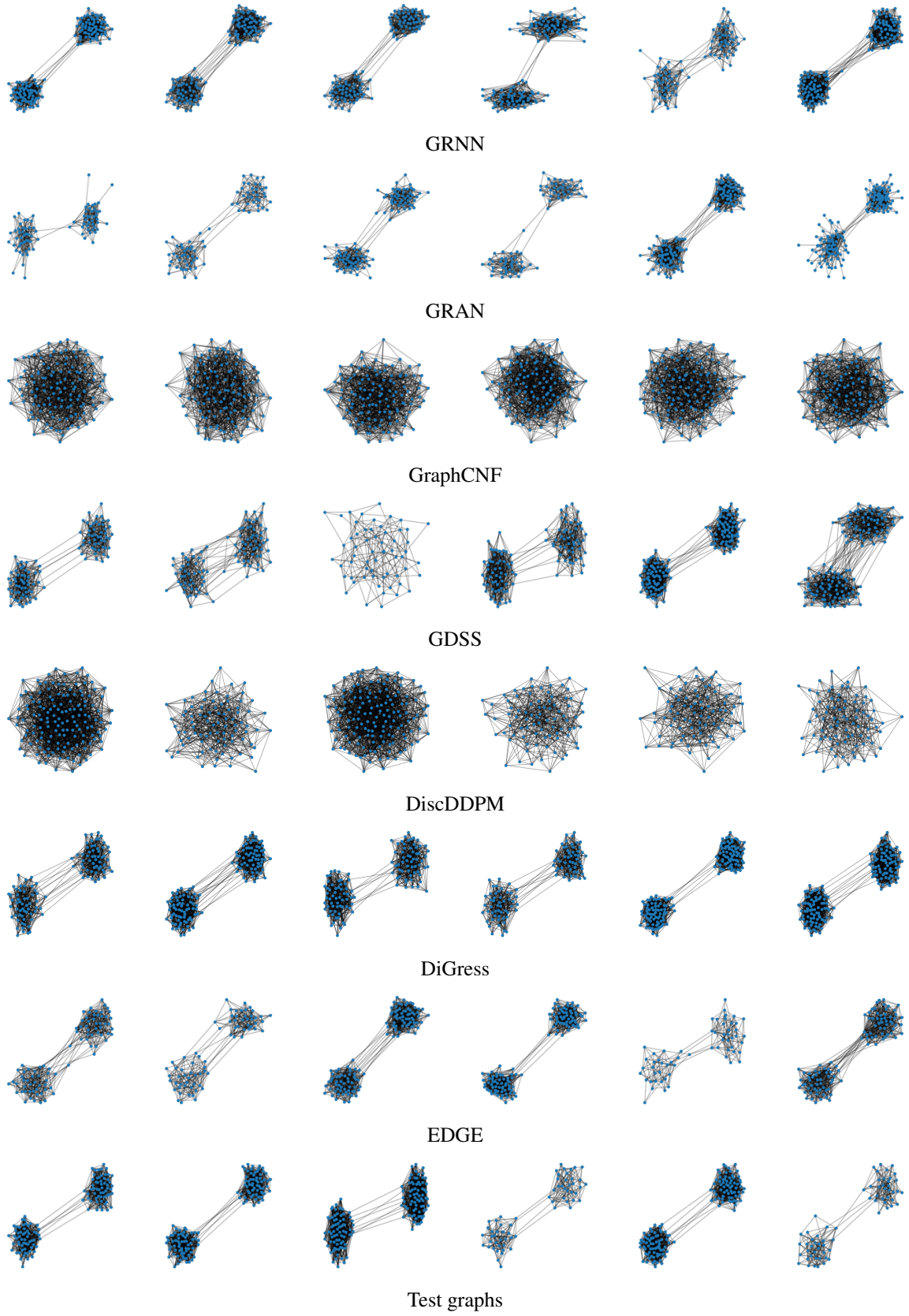


Figure 8. Visualization of the Community dataset

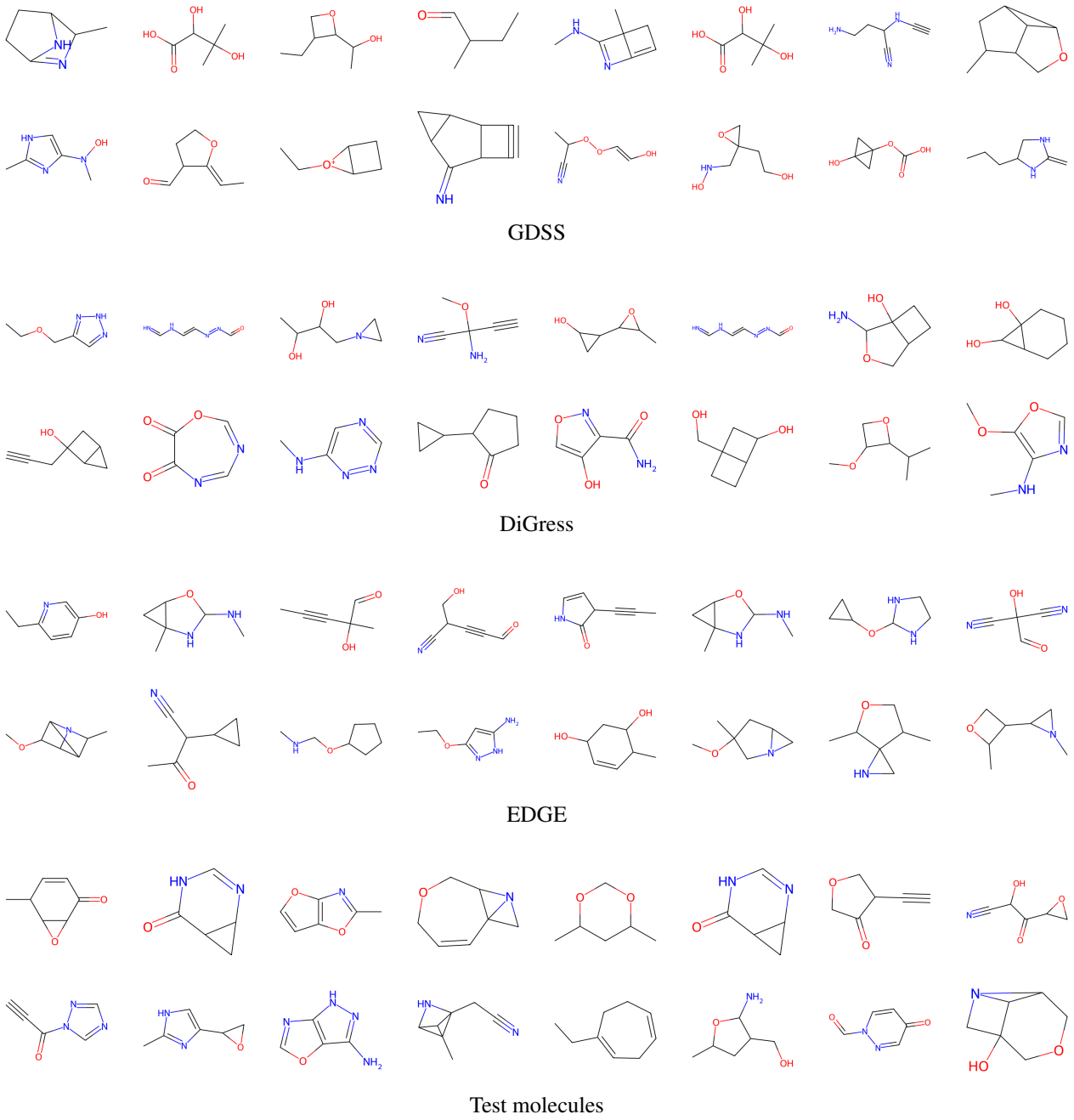


Figure 9. Visualization of the QM9 dataset