Abstract

Denoising diffusion probabilistic models and score matching models have proven to be very powerful for generative tasks. While these approaches have also been applied to the generation of discrete graphs, they have, so far, relied on continuous Gaussian perturbations. Instead, in this work, we suggest using discrete noise for the forward Markov process. This ensures that in every intermediate step the graph remains discrete. Compared to the previous approach, our experimental results on four datasets and multiple architectures show that using a discrete noising process results in higher quality generated samples indicated with an average MMDs reduced by a factor of 1.5. Furthermore, the number of denoising steps is reduced from 1000 to 32 steps leading to a 30 times faster sampling procedure.

1 Introduction

Score-based \cite{1} and denoising diffusion probabilistic models (DDPMs) \cite{2,3} have recently achieved striking results in generative modeling and in particular in image generation. Instead of learning a complex model that generates samples in a single pass (like a Generative Adversarial Network \cite{4} (GAN) or a Variational Auto-Encoder \cite{5} (VAE)), a diffusion model is a parameterized Markov Chain trained to reverse an iterative predefined process that gradually transforms a sample into pure noise. Although diffusion processes have been proposed for both continuous \cite{6} and discrete \cite{7} state spaces, their use for graph generation has only focused on Gaussian diffusion processes which operate in the continuous state space \cite{8,9}.

This contribution suggests adapting the denoising procedure to an actual graph distribution and using discrete noise, leading to a random graph model. We describe this procedure based on the Discrete DDPM framework proposed by Austin et al. \cite{7}, Hoogeboom et al. \cite{10}. Our experiments show that using discrete noise greatly reduces the number of denoising steps that are needed and improves the sample quality. We also suggest the use of a simple expressive graph neural network architecture \cite{11} for denoising, which, while bringing expressivity benefits, contrasts with more complicated architectures currently used for graph denoising \cite{8}.

2 Related Work

Traditionally, graph generation has been studied through the lens of random graph models \cite{12,14}. While this approach is insufficient to model many real-world graph distributions, it is useful to create synthetic datasets and provides a useful abstraction. In fact, we will use Erdős–Rényi (ER) graphs \cite{12} to model the prior distribution of our diffusion process.

Due to their expressive power, deep generative models have achieved better results in modeling complex graph distributions. The most successful graph generative models can be divided into two
camps: a) auto-regressive graph generative models, which generate the graph sequentially node-by-node \cite{15,16}, and b) one-shot generative models which generate the whole graph in a single forward pass \cite{17,20,8,9,21}. While auto-regressive models can generate graphs with hundreds or even thousands of nodes, they can suffer from mode collapse \cite{20,21}. One-shot graph generative models are more resilient to mode collapse but are more challenging to train while still not scaling easily beyond tens of nodes. Recently, one-shot generation has been scaled up to graphs of hundreds of nodes thanks to spectral conditioning \cite{21}, suggesting that good conditioning can largely benefit graph generation. Still, the suggested training procedure is cumbersome as it involves 3 different intertwined Generative Adversarial Networks (GANs). Finally, Variational Auto Encoders (VAE) have also been studied to generate graphs but remain difficult to train, as the loss function needs to be permutation invariant \cite{22} which can necessitate an expensive graph matching step \cite{17}.

In contrast, the score-based models \cite{8,9} have the potential to provide both, a simple, stable training objective similar to the auto-regressive models and good graph distribution coverage provided by the one-shot models. Niu et al. \cite{8} provided the first score-based model for graph generation by directly using the score-based model formulation of Song and Ermon \cite{11} and additionally accounting for the permutation equivariance of graphs. Jo et al. \cite{9} extended this to featured graph generation, by formulating the problem as a system of two stochastic differential equations, one for feature generation and one for adjacency generation. The graph and the features are then generated in parallel. This approach provided promising results for small molecule generation. Importantly, both contributions rely on a continuous Gaussian noise process and use a thousand denoising steps to achieve good results, which makes for a slow graph generation.

As shown by Song et al. \cite{6}, score matching is tightly related to denoising diffusion probabilistic models \cite{3} which provide a more flexible formulation, more easily amendable for the graph generation. In particular, for the noisy samples to remain discrete graphs, the perturbations need to be discrete.

Such discrete diffusion has been successfully used for quantized image generation \cite{23,24} and text generation \cite{25}. Diffusion using the multinomial distribution was proposed in Hoogeboom et al. \cite{10}. Then, Austin et al. \cite{7} extended the previous work by Hoogeboom et al. \cite{10}. Song et al. \cite{26} and provided a general recipe for denoising diffusion models in discrete state-spaces which mainly requires the specification of a doubly-stochastic Markov transition matrix $Q$ which ensures the Markov process conserves probability mass and converges to a stationary distribution. In the next section, we describe a formulation of this perturbation matrix $Q$ leading to the ER random graphs.

3 Discrete Diffusion for Simple Graphs

Diffusion models \cite{2} are generative models based on a forward and a reverse Markov process.

The forward process, denoted $q(A_{1:T} \mid A_0) = \prod_{t=1}^{T} q(A_t \mid A_{t-1})$ generates a sequence of increasingly noisier latent variables $A_t$ from the initial sample $A_0$, to white noise $A_T$. Here the sample $A_0$ and the latent variables $A_t$ are adjacency matrices. The learned reverse process $p_\theta(A_{1:T}) = p(A_T) \prod_{t=1}^{T} q(A_{t-1} \mid A_t)$ attempts to progressively denoise the latent variable $A_t$ in order to produce samples from the desired distribution. Here we will focus on simple graphs, but the approach can be extended in a straightforward manner to account for different edge types. We use the model from \cite{10} and, for convenience, adopt the representation of \cite{7} for our discrete process.

3.1 Forward Process

Let the row vector $a^{ij}_t \in \{0, 1\}^2$ be the one-hot encoding of $i, j$ element of the adjacency matrix $A_t$. Here $t \in [0, T]$ denotes the timestep of the process, where $A_0$ is a sample from the data distribution and $A_T$ is an ER random graph. The forward process is described as repeated multiplication of each adjacency element type row vector $a^{ij}_t = a^{ij}_{t-1} Q_t$ with a double stochastic matrix $Q_t$. Note that the forward process is independent for each edge/non-edge $i \neq j$. The matrix $Q_t \in \mathbb{R}^{2 \times 2}$ is modeled as

$$Q_t = \begin{bmatrix} 1 - \beta_t & \beta_t \\ \beta_t & 1 - \beta_t \end{bmatrix} ,$$

where $\beta_t$ is the probability of not changing the edge state. This formulation has the advantage to allow direct sampling at any timestep of the diffusion process without computing any previous timesteps. Indeed the matrix $Q_t = \prod_{1 \leq i < t} Q_i$ can be expressed in the form of \cite{1} with $\beta_t$ being
3.2 Reverse Process

To sample from the data distribution, the forward process needs to be reversed. Therefore, we need to estimate \( q(A_{t-1} | A_t, A_0) \). In our case, using the Markov property of the forward process this can be rewritten as (see Appendix A for derivation):

\[
q(A_{t-1} | A_t, A_0) = q(A_t | A_{t-1}) q(A_{t-1} | A_0) / q(A_t | A_0).
\]

(2)

Note that (2) is entirely defined by \( \beta_t \) and \( \tilde{\beta}_t \) and \( A_0 \) (see Appendix A Equation 4).

3.3 Loss

Diffusion models are typically trained to minimize a variational upper bound on the negative log-likelihood. This bound can be expressed as (see Appendix C or [3, Equation 5]):

\[
L_{vb}(A_0) := \mathbb{E}_{q(A_0)} \left[ \frac{D_{KL}(q(A_T | A_0)||p_\theta(A_T))}{L_T} \right] \\
+ \sum_{t=1}^{T} \mathbb{E}_{q(A_t | A_0)} \frac{D_{KL}(q(A_{t-1} | A_t, A_0)||p_\theta(A_{t-1} | A_t))}{L_t} - \mathbb{E}_{q(A_t | A_0)} \log(p_\theta(A_0 | A_1)) \left[ \frac{1}{L_0} \right]
\]

Practically, the model is trained to directly minimize the losses \( L_t \), i.e. the KL divergence \( D_{KL}(q(A_{t-1} | A_t, A_0)||p_\theta(A_{t-1} | A_t)) \) by using the tractable parametrization of \( q(A_{t-1} | A_t, A_0) \) from (2). Note that the discrete setting of the selected noise distribution prevents training the model to approximate the gradient of the distribution as done by score-matching graph generative models [6, 9].

Parametrization of the reverse process. While it is possible to predict the logits of \( p_\theta(A_{t-1} | A_t) \) in order to minimize \( L_{vb} \), we follow [3, 10] and use a network \( nn_q(A_t) \) that predict the logits of the distribution \( p_\theta(A_0 | A_t) \). This parametrization is known to stabilize the training procedure. To minimize \( L_{vb} \), (3) can be used to recover \( p_\theta(A_{t-1} | A_t) \) from \( A_0 \) and \( A_t \).

Alternate loss. Many implementations of DDPMs found it beneficial to use alternative losses. For instance, [3] derived a simplified loss function that reweights the ELBO. Hybrid losses have been used in [27] and [2]. As shown in Appendix D using the parametrization \( p_\theta(A_0 | A_t) \), one can express the term: \( L_t \) as \( L_t = - \log(p_\theta(A_0 | A_t)) \). Empirically, we found that minimizing

\[
L_{\text{simple}} := -\mathbb{E}_{q(A_0)} \sum_{t=1}^{T} \left( 1 - 2 \cdot \beta_t + \frac{1}{T} \right) \cdot \mathbb{E}_{q(A_t | A_0)} \log(p_\theta(A_0 | A_t))
\]

(3)

leads to stable training and better results. Note that this loss equals the cross-entropy loss between \( A_0 \) and \( nn_q(A_t) \). The re-weighting \( 1 - 2 \cdot \beta_t + \frac{1}{T} \) which assigns linearly more importance to the less noisy samples, has been proposed in [27, Equation 7].

3.4 Sampling

For each loss, we used a specific sampling algorithm. For both approaches, we start by sampling each edge independently from a Bernoulli distribution with probability \( p = 1/2 \) (ER random graph). Then, for the \( L_{vb} \) loss we follow Ho et al. [3] and iteratively reverse the chain by sampling Bernoulli-sampling from \( p_\theta(A_{t-1} | A_t) \) until we obtain at our sample of \( p_\theta(A_0 | A_t) \). For the loss function

\( L_{\text{simple}} \), we sample \( A_0 \) directly from \( p_\theta(A_0 | A_t) \) for each step \( t \) and obtain \( A_{t-1} \) by sampling again from \( q(A_{t-1} | A_0) \). The two approaches are described algorithmically in Appendix E.
The values of $\beta_t$ are selected following a simple linear schedule for our reverse process \[2\]. We found it works similarly well as other options such as cosine schedule \[27\]. Note that in this case $\beta_t$ can be obtained from $\beta_t$ in a straightforward manner (see Appendix \[B\]).

4 Experiments

We compare our graph discrete diffusion approach to the original score-based approach proposed by Niu et al. \[8\]. Models using this original formulation are denoted by score. We follow the training and evaluation setup used by previous contributions \[15, 19, 8, 9\]. More details can be found in Appendix \[C\]. For evaluation, we compute MMD metrics from \[15\] between the generated graphs and the test set, namely, the degree distribution, the clustering coefficient, and the 4-node orbit counts. To demonstrate the efficiency of the discrete parameterization, the discrete models only use 32 denoising steps, while the score-based models use 1000 denoising steps, as originally proposed. We compare two architectures: 1. EDP-GNN as introduced by Niu et al. \[8\], and 2. a simpler and more expressive provably powerful graph network (PPGN) \[11\]. See Appendix \[D\] for a more detailed description of the architectures.

Table \[1\] shows the results for two datasets, Community-small ($12 \leq n \leq 20$) and Ego-small ($4 \leq n \leq 18$), used by Niu et al. \[8\]. To better compare our approach to traditional score-based graph generation, in Table \[2\] we additionally perform experiments on slightly more challenging datasets with larger graphs. Namely, a stochastic-block-model (SBM) dataset with three communities, which in total consists of ($24 \leq n \leq 27$) nodes and a planar dataset with ($n = 60$) nodes. Detailed information on the datasets can be found in Appendix \[H\]. Additional details concerning the evaluation setup are provided in Appendix \[G.4\].

**Table 1:** MMD results for the Community and the Ego datasets. All values are averaged over 5 runs with 1024 generated samples without any sub-selection. The "Total" column denotes the average MMD over all of the 6 measurements. The best results of the "Avg." and "Total" columns are shown in bold. † marks the results taken from the original papers.

<table>
<thead>
<tr>
<th>Model</th>
<th>Community</th>
<th>Ego</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Deg</td>
<td>Clus</td>
</tr>
<tr>
<td>GraphRNN†</td>
<td>0.030</td>
<td>0.030</td>
</tr>
<tr>
<td>GN†</td>
<td>0.120</td>
<td>0.150</td>
</tr>
<tr>
<td>EDP-Score†</td>
<td>0.086</td>
<td>0.127</td>
</tr>
<tr>
<td>SDE-Score†</td>
<td>0.045</td>
<td>0.086</td>
</tr>
<tr>
<td>EDP-Score</td>
<td>0.016</td>
<td>0.010</td>
</tr>
<tr>
<td>PPGN-Score</td>
<td>0.081</td>
<td>0.237</td>
</tr>
<tr>
<td>PPGN $L_{vb}$</td>
<td>0.023</td>
<td>0.064</td>
</tr>
<tr>
<td>PPGN $L_{simple}$</td>
<td>0.019</td>
<td>0.044</td>
</tr>
<tr>
<td>EDP $L_{vb}$</td>
<td>0.024</td>
<td>0.04</td>
</tr>
</tbody>
</table>

**Table 2:** MMD results for the SBM-27 and the Planar-60 datasets.

<table>
<thead>
<tr>
<th>Model</th>
<th>SBM-27</th>
<th>Planar-60</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Deg</td>
<td>Clus</td>
</tr>
<tr>
<td>EDP-Score</td>
<td>0.054</td>
<td>0.800</td>
</tr>
<tr>
<td>PPGN $L_{simple}$</td>
<td>0.007</td>
<td>0.035</td>
</tr>
<tr>
<td>EDP $L_{simple}$</td>
<td>0.046</td>
<td>0.184</td>
</tr>
</tbody>
</table>

**Results.** In Table \[1\] we observe that the proposed discrete diffusion process using the $L_{vb}$ loss and PPGN model leads to slightly improved average MMDs over the competitors. The $L_{simple}$ loss further improves the result over $L_{vb}$. The fact that the EDP-$L_{simple}$ model has significantly lower MMD values than the EDP-score model is a strong indication that the proposed loss and the discrete formulation are the cause of the improvement rather than the PPGN architecture. This improvement comes with the additional benefit that sampling is greatly accelerated (30 times) as the number of timesteps is reduced from 1000 to 32. Table \[2\] shows that the proposed discrete formulation is even more beneficial when graph size and complexity increase. The PPGN-Score even becomes infeasible to run in this setting, due to the prohibitively expensive sampling procedure. A qualitative evaluation of the generated graphs is performed in Appendix \[I\]. Visually, the $L_{simple}$ loss leads to the best samples.

5 Conclusion

In this work, we demonstrated that discrete diffusion can increase sample quality and greatly improve the efficiency of denoising diffusion for graph generation. While the approach was presented for simple graphs with non-attributed edges, it could also be extended to graphs with edge attributes.

\[1\] The discrepancy with the SDE-Score results comes from the fact that using the code provided by the authors, we were unable to reproduce their results. Strangely, their code leads to good results when used with our discrete formulation and $L_{simple}$ loss improving over the result reported in their contribution.
References


A Reverse Process Derivations

In this appendix, we provide the derivation of the reverse probability $q(A_{t-1}|A_t, A_0)$. Using the Bayes rule, we obtain

$$q(A_{t-1}|A_t, A_0) = \frac{q(A_t | A_{t-1}, A_0) \cdot q(A_{t-1} | A_0)}{q(A_t | A_0)}$$

$$= \frac{q(A_t | A_{t-1}) \cdot q(A_{t-1} | A_0)q(A_0)}{q(A_t | A_0)}$$

$$= \frac{q(A_t | A_{t-1}) \cdot q(A_{t-1} | A_0)}{q(A_t | A_0)},$$

where we use the fact that $q(A_t | A_{t-1}, A_0) = q(A_t | A_{t-1})$ since $A_t$ is independent of $A_0$ given $A_{t-1}$.

This reverse probability is entirely defined with $\beta_t$ and $\bar{\beta}_t$. For the $i, j$ element of $A$ (denoted $A^{ij}$), we obtain:

$$q(A^{ij}_{t-1} = 1 | A^{ij}_t, A^{ij}_0) = \begin{cases} 
    (1 - \beta_t) \cdot \frac{(1-\bar{\beta}_{t-1})}{1-\bar{\beta}_t}, & \text{if } A^{ij}_t = 1, A^{ij}_0 = 1 \\
    (1 - \beta_t) \cdot \frac{\bar{\beta}_{t-1}}{\beta_t}, & \text{if } A^{ij}_t = 1, A^{ij}_0 = 0 \\
    \beta_t \cdot \frac{1-\bar{\beta}_{t-1}}{1-\beta_t}, & \text{if } A^{ij}_t = 0, A^{ij}_0 = 1 \\
    \beta_t \cdot \frac{\bar{\beta}_{t-1}}{1-\beta_t}, & \text{if } A^{ij}_t = 0, A^{ij}_0 = 0 
\end{cases}$$

B Conversion of $\bar{\beta}_t$ to $\beta_t$

The selected linear schedule provides us with the values of $\bar{\beta}_t$. In this appendix, we compute an expression for $\beta_t$ from $\bar{\beta}_t$, which allows us easy computation of $q$. By definition, we have $\bar{Q}_t = \bar{Q}_{t-1}Q_t$ which is equivalent to

$$\left( \begin{array}{cc} 1 - \bar{\beta}_{t-1} & \bar{\beta}_{t-1} \\ \bar{\beta}_{t-1} & 1 - \bar{\beta}_{t-1} \end{array} \right) \left( \begin{array}{cc} 1 - \beta_t & \beta_t \\ \beta_t & 1 - \beta_t \end{array} \right) = \left( \begin{array}{cc} 1 - \bar{\beta}_t & \bar{\beta}_t \\ \bar{\beta}_t & 1 - \bar{\beta}_t \end{array} \right)$$

Let us select the first row and first column equality. We obtain the following equation

$$(1 - \beta_{t-1}) (1 - \beta_t) + \bar{\beta}_{t-1} \beta_t = 1 - \bar{\beta}_t,$$

which, after some arithmetic, provides us with the desired answer

$$\beta_t = \frac{\bar{\beta}_{t-1} - \bar{\beta}_t}{2\bar{\beta}_{t-1} - 1}.$$
We note that this term corresponds to the cross-entropy of the distribution \(p\). We use \(L\) where (5) follows from

\[
L_{vb} = \mathbb{E}_{q(A_{0:T})} \left[ - \log \left( \frac{p_\theta(A_{0:T})}{q(A_{1:T} \mid A_0)} \right) \right]
\]

\[
= \mathbb{E}_q \left[ - \log (p_\theta(A_T)) - \sum_{t=1}^T \log \left( \frac{p_\theta(A_{t-1} \mid A_t)}{q(A_t \mid A_{t-1})} \right) \right]
\]

\[
= \mathbb{E}_q \left[ - \log (p_\theta(A_T)) - \sum_{t=2}^T \log \left( \frac{p_\theta(A_{t-1} \mid A_t)}{q(A_t \mid A_{t-1})} \right) - \log \left( \frac{p_\theta(A_0 \mid A_1)}{q(A_1 \mid A_0)} \right) \right]
\]

\[
= \mathbb{E}_q \left[ - \log (p_\theta(A_T)) - \sum_{t=2}^T \log \left( \frac{p_\theta(A_{t-1} \mid A_t)}{q(A_t \mid A_{t-1})} \right) - \log (p_\theta(A_0 \mid A_1)) \right]
\]

\[
= \mathbb{E}_{q(A_{0:T})} \left[ D_{KL}(q(A_{t-1} \mid A_t, A_0) \parallel p_\theta(A_{t-1} \mid A_t)) \right] + \sum_{t=2}^T \mathbb{E}_{q(A_{t-1} \mid A_t, A_0)} D_{KL}(q(A_{t-1} \mid A_t, A_0) \parallel p_\theta(A_{t-1} \mid A_t))
\]

\[-\mathbb{E}_{q(A_{1} \mid A_0)} \log(p_\theta(A_0 \mid A_1))\]

where (5) follows from

\[
q(A_{t-1} \mid A_t, A_0) = \frac{q(A_{t-1}, A_0)}{q(A_t \mid A_0)} = \frac{q(A_{t-1}, A_0)}{q(A_t \mid A_0)} \cdot \frac{q(A_t \mid A_{t-1}, A_0)}{q(A_t \mid A_0)}.
\]

### D Simple Loss

Using the parametrization \(p_\theta(A_0 \mid A_t)\), we can simplify the KL divergence of the term \(L_t\).

\[
D_{KL}(q(A_{t-1} \mid A_t, A_0) \parallel p_\theta(A_{t-1} \mid A_t)) = \mathbb{E}_{q(A_{t-1} \mid A_t, A_0)} \left[ - \log \left( \frac{p_\theta(A_{t-1} \mid A_t)}{q(A_{t-1} \mid A_t, A_0)} \right) \right]
\]

\[
= \mathbb{E}_{q(A_{t-1} \mid A_t, A_0)} \left[ - \log (p_\theta(A_0 \mid A_t)) \right]
\]

\[-\mathbb{E}_{q(A_{1} \mid A_0)} \log(p_\theta(A_0 \mid A_1))\]

We note that this term corresponds to the cross-entropy of the distribution \(p_\theta(A_0 \mid A_t\) with the ground truth of \(A_0\).

### E Sampling Algorithms

Here in Algorithms 1 and 2 we provide an algorithmic description of the two sampling approaches described in Section 3.4. Here \(B_{p=1/2}\) denotes the Bernoulli distribution with parameter \(p = 1/2\), which corresponds to the Erdős–Rényi random graph model.

**Algorithm 1** Sampling for \(L_{vb}\)

1: \(\forall i, j \mid i > j: A_{ij}^T \sim B_{p=1/2}\)
2: for \(t = T, \ldots, 1\) do
3: Compute \(p_\theta(A_{t-1} \mid A_t)\)
4: \(A_{t-1} \sim p_\theta(A_{t-1} \mid A_t)\)
5: end for

**Algorithm 2** Sampling for \(L_{simple}\)

1: \(\forall i, j \mid i > j: A_{ij}^T \sim B_{p=1/2}\)
2: for \(t = T, \ldots, 1\) do
3: \(\tilde{A}_0 \sim p_\theta(A_0 \mid A_t)\)
4: \(A_{t-1} \sim q(A_{t-1} \mid \tilde{A}_0)\)
5: end for
F Models

F.1 Edgewise Dense Prediction Graph Neural Network (EDP-GNN)

The EDP-GNN model introduced by Niu et al. [8] extends GIN [28] to work with multi-channel adjacency matrices. This means that a GIN graph neural network is run on multiple different adjacency matrices (channels) and the different outputs are concatenated to produce new node embeddings:

\[ X^{(k+1)'} = \overline{A}^{(k)} X^{(k)} + (1 + \epsilon) X^{(k)}, \]

\[ X^{(k+1)} = \text{Concat}(X^{(k+1)'}, \text{for } c \in \{1, \ldots, C^{(k+1)}\}), \]

where \( X \in \mathbb{R}^{n \times h} \) is the node embedding matrix with hidden dimension \( h \) and \( C^{(k)} \) is the number of channels in the input multi-channel adjacency matrix \( A^{(k)} \in \mathbb{R}^{C^{(k)} \times n \times n} \), at layer \( k \). The adjacency matrices for the next layer are produced using the node embeddings:

\[ \overline{A}^{(k+1)}_{i,j} = \text{MLP}(\overline{A}^{(k)}_{i,j}, X_{i}, X_{j}). \]

For the first layer, EDP-GNN computes two adjacency matrix \( \overline{A}^{(0)} \) channels, original input adjacency \( A \) and its inversion \( 11^T - A \). For node features, node degrees are used \( X^{(0)} = \sum_{i} A_{i} \).

To produce the final outputs, outputs of all intermediary layers are concatenated:

\[ \overline{A} = \text{MLP}_{out}(\text{Concat}(\overline{A}^{(k)} \text{ for } k \in \{1, \ldots, K\})). \]

The final layer always has only one output channel, such that \( A_{(0)} = \text{EDP-GNN}(A_{(t-1)}) \).

To condition the model on the given noise level \( \beta \), noise-level-dependent scale and bias parameters \( \alpha_t \) and \( \gamma_t \) are introduced to each layer \( f \) of every MLP:

\[ f(\overline{A}_{i,j}) = \text{activation}(W\overline{A}_{i,j} + b)\alpha_t + \gamma_t. \]

F.2 Provably Powerful Graph Network (PPGN)

The input to the PPGN model used is the adjacency matrix \( A \), concatenated with the diagonal matrix \( \overline{\beta} \cdot I \), resulting in an input tensor \( X_{in} \in \mathbb{R}^{n \times n \times 2} \). The output tensor is \( X_{out} \in \mathbb{R}^{n \times n \times 1} \), where each \( [X_{out}]_{ij} \) represents \( p(\overline{A}_{ij} \mid |A_{ij}|) \).

Our PPGN implementation, which closely follows Maron et al. [11], is structured as follows:

Let \( P \) denote the PPGN model, then

\[ P(X_{in}) := (l_{out} \circ C)(X_{in}) \]

\[ C : \mathbb{R}^{n \times n \times 2} \rightarrow \mathbb{R}^{n \times n \times (d \cdot h)} \]

\[ C(X_{in}) := \text{Concat}((B_d \circ \ldots \circ B_1)(X_{in}),(B_{d-1} \circ \ldots \circ B_1)(X_{in}), \ldots, B_1(X_{in})) \]

The set \( \{B_1, \ldots, B_d\} \) is a set of \( d \) different powerful layers implemented as proposed by Maron et al. [11]. We let the input run through different amounts of these powerful layers and concatenate their respective outputs to one tensor of size \( n \times n \times (d \cdot h) \). These powerful layers are functions of size:

\[ \forall B_i \in \{B_2, \ldots, B_d\}, B_i : \mathbb{R}^{n \times n \times h} \rightarrow \mathbb{R}^{n \times n \times h} \]

\[ B_1 : \mathbb{R}^{n \times n \times 1} \rightarrow \mathbb{R}^{n \times n \times h} \]

Finally, we use an MLP 2 to reduce the dimensionality of each matrix element down to 1, so that we can treat the output as an adjacency matrix.

\[ l_{out} : \mathbb{R}^{d \cdot h} \rightarrow \mathbb{R}^{1}, \]

where \( l_{out} \) is applied to each element \( [C(X_{in})]_{i,j} \) of the tensor \( C(X_{in}) \) over all its \( d \cdot h \) channels. It is used to reduce the number of channels down to a single one which represents \( p(A_{ij} \mid A_t) \).
G Training Setup

G.1 EDP-GNN

The model training setup and hyperparameters used for the EDP-GNN were directly taken from [8].
We used 4 message-passing steps for each GIN, then stacked 5 EDP-GNN layers, for which the
maximum number of channels is always set to 4 and the maximum number of node features is 16.
We use 32 denoising steps for all datasets besides Planar-60, where we used 256. Opposed to 6 noise
levels with 1000 sample steps per level as in the Score-based approach.

G.2 PPGN

The PPGN model we used for the Ego-small, Community-small, and SBM-27 datasets consist of
6 layers \( \{ B_1, \ldots, B_6 \} \). After each powerful layer, we apply an instance normalization. The hidden
dimension was set to 16. For the Planar-60 dataset, we have used 8 layers and a hidden dimension of
128. We used a batch size of 64 for all datasets and used the Adam optimizer with parameters chosen
as follows: learning rate is 0.001, betas are \((0.9, 0.999)\) and weight decay is 0.999.

G.3 Model Selection

We performed a simple model selection where the model which achieves the best training loss is
saved and used to generate graphs for testing. We also investigated the use of a validation split and
computation of MMD scores versus this validation split for model selection, but we did not find this
to produce better results while adding considerable computational overhead.

G.4 Additional Details on Experimental Setup

Here we provide some details concerning the experimental setup for the results in Tables 1 and 2.

Details for MMD results in Table 1: From the original paper Niu et al. [8], we are unsure if the
GNF, GraphRNN, and EDP-Score model selection were used or not. The SDE-Score results in the
first section are sampled after training for 5000 epochs and no model selection was used. Due to the
compute limitations on the PPGN model, the results for PPGN \( L_{vb} \) are taken after epoch 900 instead
of 5000, as results for SDE-Score and EDP-Score have been. The results for PPGN \( L_{simple} \) and EDP
\( L_{simple} \) were trained for 2500 epochs.

Details for MMD results in Table 2: All results using the EDP-GNN model are trained until epoch
5000 and the PPGN implementation was trained until epoch 2500.

H Datasets

In this appendix, we describe the 4 datasets used in our experiments.

Ego-small: This dataset is composed of 200 graphs of 4-18 nodes from the Citeseer network (Sen
et al. [29]). The dataset is available in the repository\(^1\) of Niu et al. [8].

Community-small: This dataset consists of 100 graphs from 12 to 20 nodes. The graphs are
generated in two steps. First two communities of equal size are generated using the Erdos-Rényi
model [12] with parameter \( p = 0.7 \). Then edges are randomly added between the nodes of the two
communities with a probability \( p = 0.05 \). The dataset is directly taken from the repository of Niu
et al. [8].

SBM-27: This dataset consists of 200 graphs with 24 to 27 nodes generated using the Stochastic-
Block-Model (SBM) with three communities. We use the implementation provided by Martinkus
et al. [21]. The parameters used are \( p_{intra} = 0.85, p_{inter} = 0.046875 \), where \( p_{intra} \) stands for the
intra-community (i.e. for node within the same community) edge probability and \( p_{inter} \) stands for the

\(^1\)https://github.com/ermongroup/GraphScoreMatching
inter-community (i.e. for nodes from different community) edge probability. The number of nodes for the 3 communities is randomly drawn from \( \{7, 8, 9\} \). In expectation, these parameters generate 3 edges between each pair of communities.

**Planar-60:** This dataset consists of 200 randomly generated planar graphs of 60 nodes. We use the implementation provided by Martinkus et al. [21]. To generate a graph, 60 points are first random uniformly sampled on the \([0, 1]^2\) plane. Then the graph is generated by applying Delaunay triangulation to these points [30].

I Visualization of Sampled Graphs

In the following pages, we provide a visual comparison of graphs generated by the different models.
Figure 1: Sample graphs from the training set of Ego-small dataset.

Figure 2: Sample graphs generated with the model EDP-Score [8] for the Ego-small dataset.

Figure 3: Sample graphs generated with the PPGN $L_{vb}$ model for the Ego-small dataset.

Figure 4: Sample graphs generated with the EDP $L_{simple}$ model for the Ego-small dataset.
Figure 5: Sample graphs from the training set of the Community dataset.

Figure 6: Sample graphs generated with the model EDP-Score \[ 8 \] for the Community dataset.

Figure 7: Sample graphs generated with the PPGN \( L_{vb} \) model for the Community dataset.

Figure 8: Sample graphs generated with the EDP \( L_{simple} \) model for the Community dataset.
Figure 9: Sample graphs from the training set of the Planar-60 dataset.

Figure 10: Sample graphs generated with the model EDP-Score [8] for the Planar-60 dataset.

Figure 11: Sample graphs generated with the PPGN $L_{\text{simple}}$ model for the Planar-60 dataset.
Figure 12: Sample graphs from the training set of the SBM-27 dataset.

Figure 13: Sample graphs generated with the model EDP-Score [8] for the SBM-27 dataset.

Figure 14: Sample graphs generated with the PPGN $L_{simple}$ model for the SBM-27 dataset.