Diffusion Model for graph generation

Tianyu Xie
2022/11/12

1. Graph Genenration via Score-Based Generative Modeling

Score-Based Graph Generation
Problem Setting

- Score-based generative modeling.
- The training loss is defined using DSM

$$\sum_{i=1}^{L} \frac{\sigma_i^2}{2L} \mathbb{E}\left[\| \mathbf{s}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}, \sigma_i) - \nabla_{\tilde{\mathbf{x}}} \log q_{\sigma_i}(\tilde{\mathbf{x}} \mid \mathbf{x}) \|_2^2 \right].$$

- The sampling process follows annealed Langevin dynamics sampling.

Algorithm 1 Annealed Langevin dynamics sampling.

```
Require: $\{\sigma_i\}_{i=1}^L, \epsilon, T$   ▷ $\epsilon$ is smallest step size; $T$ is
    the number of iteration for each noise level.
 1: Initialize $\tilde{\mathbf{x}}_0$
 2: for $i \leftarrow 1$ to $L$ do
 3:     $\alpha_i \leftarrow \epsilon \cdot \sigma_i^2 / \sigma_L^2$        ▷ $\alpha_i$ is the step size.
 4:     for $t \leftarrow 1$ to $T$ do
 5:         Draw $\mathbf{z}_t \sim \mathcal{N}(0, I)$
 6:         $\tilde{\mathbf{x}}_t \leftarrow \tilde{\mathbf{x}}_{t-1} + \frac{\alpha_i}{2} \mathbf{s}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}_{t-1}, \sigma_i) + \sqrt{\alpha_i}\, \mathbf{z}_t$
 7:     end for
 8:     $\tilde{\mathbf{x}}_0 \leftarrow \tilde{\mathbf{x}}_T$
 9: end for
    return $\tilde{\mathbf{x}}_T$
```

Score-Based Graph Generation
Model

- How to extend score-based generation to **graph generation**?

- Let an unweighted graph be represented by its adjacency matrix $A \in \{0,1\}^{N \times N}$ where N is the number of nodes.

- The noise distribution $q_\sigma(\tilde{A} \mid A)$ is defined by

$$\begin{cases} \prod_{i<j} \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{ -\frac{(\tilde{\mathbf{A}}_{[i,j]} - \mathbf{A}_{[i,j]})^2}{2\sigma^2} \right\}, & \text{if } \tilde{\mathbf{A}} = \tilde{\mathbf{A}}^\intercal \\ 0, & \text{otherwise.} \end{cases}$$

  Intuitively, we only add Gaussian noise to the upper triangular part of the adjacency matrix.

- Since the score of noisy distribution is $\nabla_{\tilde{\mathbf{A}}} \log q_\sigma(\tilde{\mathbf{A}} \mid \mathbf{A}) = -(\tilde{\mathbf{A}} - \mathbf{A})/\sigma^2$, the DSM loss is

$$\mathcal{L}(\boldsymbol{\theta}; \{\sigma_i\}_{i=1}^L) \triangleq \frac{1}{2L} \sum_{i=1}^{L} \sigma_i^2 \mathbb{E}\left[\left\| \mathbf{s}_{\boldsymbol{\theta}}(\tilde{\mathbf{A}}, \sigma) + \frac{\tilde{\mathbf{A}} - \mathbf{A}}{\sigma^2} \right\|_2^2 \right].$$

Score-Based Graph Generation
Model

- To sample from the model, we first initialize $\tilde{A}_0$ using folded normal distributions, i.e.

  $$(\tilde{\mathbf{A}}_0)_{[i,j]} = \begin{cases} |\varepsilon_{[i,j]}|, & i < j \\ (\tilde{\mathbf{A}}_0)_{[j,i]}, & \text{otherwise,} \end{cases}$$

  where all $\epsilon \sim N(0,1)$.

- Then, we update $\tilde{A}$ by iteratively sampling from trained $s_\theta(A; \sigma_i)$ using **Langevin dynamics.** At each step, we add a minor modification by changing the noise term z_t to a symmetric one

  $$(\tilde{\mathbf{z}}_t)_{[i,j]} = \begin{cases} (\tilde{\mathbf{z}}_t)_{[i,j]}, & i < j \\ (\tilde{\mathbf{z}}_t)_{[j,i]}, & i \geq j, \end{cases}$$

  which accounts for the symmetry of adjacency matrices.

Score-Based Graph Generation
Model

- At last, we quantize the generated continuous adjacency matrix (denoted $\tilde{A}$) to a binary one (denoted by A^{sample}), by

$$\mathbf{A}_{[i,j]}^{(\text{sample})} = \mathbb{1}_{\tilde{\mathbf{A}}_{[i,j]} > 0.5}$$

Score-Based Graph Generation

Multi-Channel GNN Layer

- We introduce a GNN-based score network $s_\theta(A;\sigma)$ that can efficiently model the scores of graph distributions while being permutation equivariant.

- For the kth layer, we define a C-channel GNN layer with M message passing steps. The m-th message passing step can be expressed as follows

$$\begin{aligned}
\tilde{\mathbf{Z}}_{[c,\cdot]}^{(m+1)} &= \mathbf{A}_{[c,\cdot,\cdot]}^{(k)} \mathbf{Z}_{[\cdot]}^{(m)}, \quad \text{for } c = 0, 1, \ldots, C-1, \\
\mathbf{Z}_i^{(m+1)} &= \mathrm{MLP}_{\text{Node}}^{(m)} \Big(\mathrm{CONCAT} \Big(\\
&\quad \tilde{\mathbf{Z}}_{[c,i]}^{(m+1)} + (1+\epsilon)\mathbf{Z}_i^{(m)} | c = 0, \ldots, C-1 \Big) \Big),
\end{aligned}$$

  Here, i is the index of nodes, $A^{(k)}$ is the multichannel adjacency matrix and $Z^{(m)}$ is the vector of node features.

- Rmk: Intuitively, the first step is **message passing** between nodes, and the second step is the **aggregation** of messages.

Score-Based Graph Generation
Multi-Channel GNN Layer

- After *M* step of message passing, for each node v_i, the output feature is given by

$$(\mathbf{Z}_{\text{out}})_i = \text{CONCAT}(\mathbf{Z}_i^{(m)} | m = 0, 1, \ldots, M-1).$$

- Henceforth, we denote our Multi-Channel GNN layer as

$$\mathbf{Z}_{\text{out}} = \text{MultiChannelGNN}(\mathbf{A}, \mathbf{Z}_{\text{in}}).$$

-

Score-Based Graph Generation
EDP-GNN layer

- The Edgewise Dense Prediction Graph Neural Network (EDP-GNN) is the key conponent of our model. We want our GNN layer to **extract edgewise features and map them to a new adjacency matrix**, using local information (which is defined in terms of connectivity) of each node in the graph.

- The EDP-GNN layer has two steps:

  - Node feature inference: update the node features using the local structure of the graph

$$\mathbf{Z}^{(k+1)} = \mathrm{MultiChannelGNN}^{(k)}(\mathbf{A}^{(k)}, \mathbf{Z}^{(k)});$$

  - Edge feature inference: update the adjacency matrix using node features.

$$\tilde{\mathbf{A}}^{(k+1)}_{[\cdot,i,j]} = \mathrm{MLP}^{(k)}_{\mathrm{Edge}}\left(\mathrm{CONCAT}(\mathbf{A}^{(k)}_{[\cdot,i,j]}, \mathbf{Z}^{(k+1)}_i, \mathbf{Z}^{(k+1)}_j)\right)$$

  To ensure symmetry, the new adjacency matrix is given by

$$\mathbf{A}^{(k+1)} = \tilde{\mathbf{A}}^{(k+1)} + (\tilde{\mathbf{A}}^{(k+1)})^{\mathsf{T}}.$$

Score-Based Graph Generation

Input and Output Layer

- Input layer: Multichannel adjacency matrix and node features should be preprocessed before they are fed into out EDP-GNN model. Formally,

$$\mathbf{Z}_i^{(0)} = \sum_j \mathbf{Adj}_{[i,j]}, \forall v_i \in \mathcal{V}$$
$$\mathbf{A}_{[0,\cdot,\cdot]}^{(0)} = \mathbf{Adj}$$
$$\mathbf{A}_{[1,\cdot,\cdot]}^{(0)} = 1 - \mathbf{Adj}$$

  If we already have node features $X \in \mathbb{R}^{N \times F_0}$ from data, then we using the following initialization for each node v_i.

$$\mathbf{Z}_i^{(0)} = \mathrm{CONCAT}\left(\mathbf{X}_i, \sum_j \mathbf{Adj}_{i,j}\right).$$

- Output layer: For each edge (v_i, v_j), the output score are given by

$$\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{A})_{[i,j]} = \mathrm{MLP}_{\mathrm{final}}\left(\mathrm{CONCAT}\left(\mathbf{A}_{[\cdot,i,j]}^{(k)} | k = 0, \ldots, K-1\right)\right).$$

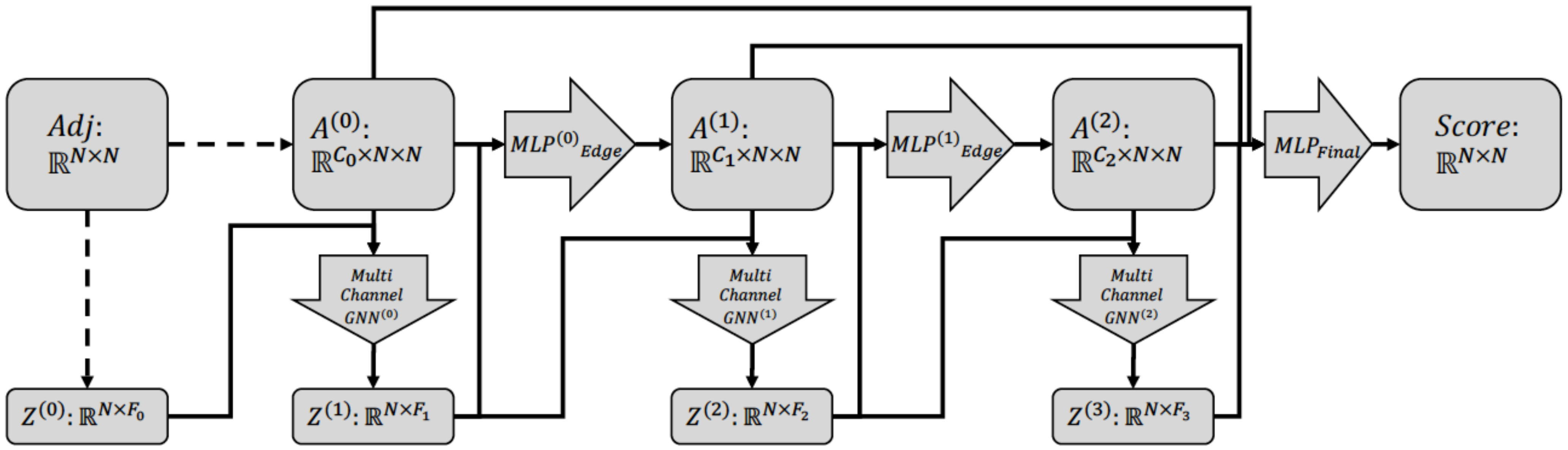
Score-Based Graph Generation
Input and Output Layer

- To enable noise level conditioning, all MLPs in EDP-GNN are substituted with conditional MLPs, that is

$$f_i(\mathbf{A}) = \mathrm{activate}((\mathbf{WA} + \mathbf{b})\boldsymbol{\alpha}_i + \boldsymbol{\beta}_i)$$

  where α_i, β_i are learnable parameters corresponding to noise level σ_i.

- This figure shows an EDP-GNN with three layers.



Adj: $\mathbb{R}^{N\times N}$
$A^{(0)}$: $\mathbb{R}^{C_0\times N\times N}$
$MLP^{(0)}{}_{Edge}$
$A^{(1)}$: $\mathbb{R}^{C_1\times N\times N}$
$MLP^{(1)}{}_{Edge}$
$A^{(2)}$: $\mathbb{R}^{C_2\times N\times N}$
MLP_{Final}
$Score$: $\mathbb{R}^{N\times N}$
Multi Channel $GNN^{(0)}$
Multi Channel $GNN^{(1)}$
Multi Channel $GNN^{(2)}$
$Z^{(0)}$: $\mathbb{R}^{N\times F_0}$
$Z^{(1)}$: $\mathbb{R}^{N\times F_1}$
$Z^{(2)}$: $\mathbb{R}^{N\times F_2}$
$Z^{(3)}$: $\mathbb{R}^{N\times F_3}$

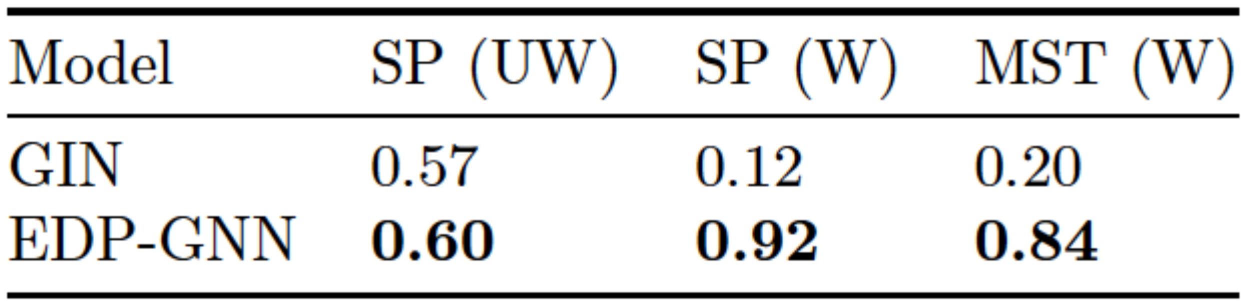
Figure 1: This figure shows an EDP-GNN with three layers. The input is an adjacency matrix of a graph with N nodes given a fixed node ordering, and the outputs are edge representations. The dashed lines are preprocessing steps, and solid lines represent network computations.

Score-Based Graph Generation
Expriments

- The first experiment tests the power of EDP-GNN. We run graph algorithm to predict whether eahch edge is in the solution set or not.
  1) Shortest Path (SP) between a given pair of nodes, including weighted and unweighted graphs.
  2) Maximum Spanning Tree (MST) of a given weighted graph.

- For boths tasks, all the graphs in the training set are randomly sampled from the Erdos and Rényi model with $n = 12$ and $p = 0.3$. For weighted graphs, all the edge weighted are uniformly sampled from [0,1].

- The test accuracy show that EDP-GNN is mode effective for edgewise predictions.



Model	SP (UW)	SP (W)	MST (W)
GIN	0.57	0.12	0.20
EDP-GNN	**0.60**	**0.92**	**0.84**

Score-Based Graph Generation

Expriments

- The second expriment demonstrate that EDP-GNN is capable of producing high-quality graph samples via score based generative modeling.

- We test the proposed model on two datasets, Community-small ($12 \leq N \leq 20$) and Ego-small ($4 \leq N \leq 18$). Our baselines include GraphRNN (You et al., 2018b), Graph Normalizing Flow(GNF) (Liu et al., 2019), GraphVAE (Simonovsky and Komodakis, 2018), and DeepGMG (Li et al., 2018a).

- We calculate the MMD of of the distribution of three graph statistics. 1) degree distribution; 2) cluster coefficient distribuition and 3) the number of orbits of 4 nodes.

Score-Based Graph Generation
Expriments

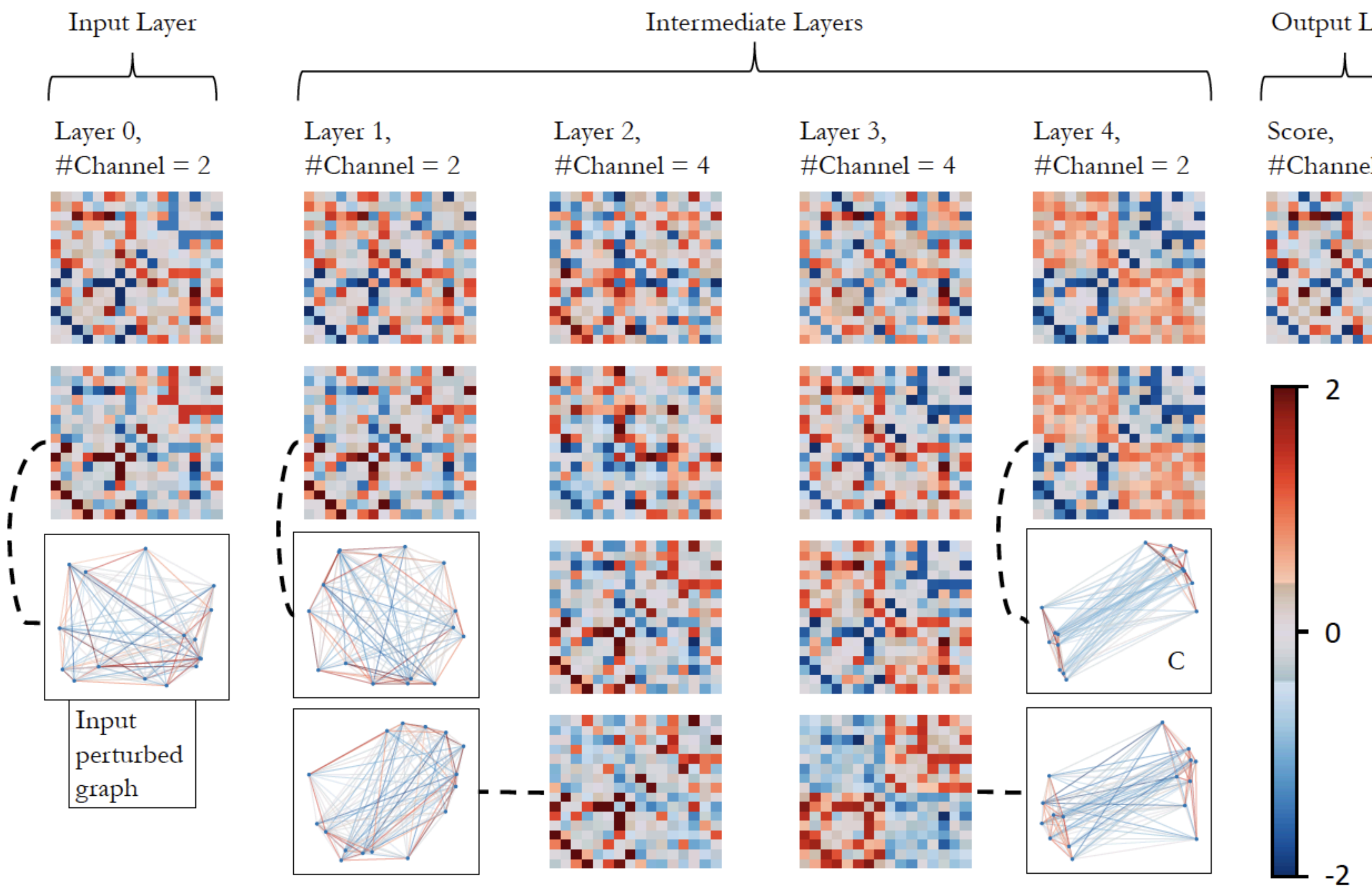


Figure 2: Visualization of channels for a pre-trained EDP-GNN model on the Community-small dataset. The model is trained with a single noise level $\sigma = 0.6$. The input is a community graph, but perturbed with Gaussian noise with $\sigma = 0.6$. The edge weights of each adjacency matrix are standardized to zero mean and unit variance. Since our model is agnostic to different permutations of nodes, we chose a specific ordering so that the adjacency matrices of community graphs possess a block diagonal form. We visualize one adjacency matrix for each layer. Sometimes a graph is less visually interpretable, and we instead visualize its complementary graph and mark it with "C". By comparing the graph visualizations for the 3rd, 4th, and the input layers, we observe that the model maps the perturbed graph with no visible structures to a graph with clear "community" structures.

Score-Based Graph Generation

Expriments

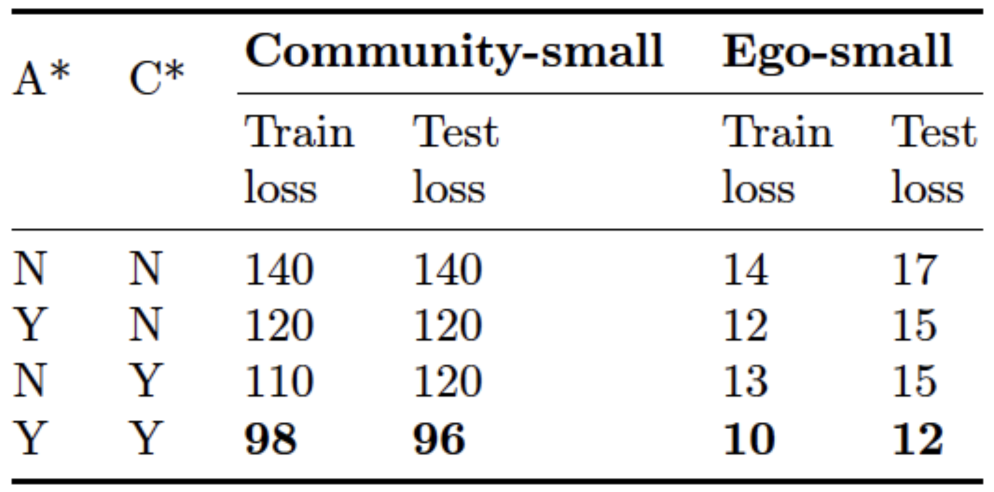
Model	Community-small				Ego-small				Avg.
	Deg.	Clus.	Orbit	Avg.	Deg.	Clus.	Orbit	Avg.	
GraphVAE	0.350	0.980	0.540	0.623	0.130	0.170	0.050	0.117	0.370
DeepGMG	0.220	0.950	0.400	0.523	0.040	0.100	0.020	0.053	0.288
GraphRNN	0.080	**0.120**	0.040	0.080	0.090	0.220	0.003	0.104	0.092
GNF	0.200	0.200	0.110	0.170	**0.030**	0.100	**0.001**	**0.044**	0.107
EDP-GNN	**0.053**	0.144	**0.026**	**0.074**	0.052	**0.093**	0.007	0.050	**0.062**
GraphRNN (1024)	0.030	**0.010**	**0.010**	**0.017**	0.040	0.050	0.060	0.050	0.033
GNF (1024)	0.120	0.150	0.020	0.097	**0.010**	0.030	**0.001**	0.014	0.055
EDP-GNN (1024)	**0.006**	0.127	0.018	0.050	**0.010**	**0.025**	0.003	**0.013**	**0.031**

Table 2: MMD results of various graph generative models. Rows marked with (1024) mean the corresponding number of samples is 1024; otherwise, the number of samples equals the size of the test set. Apart from three MMD statistics, we also provide their average values, noted as "Avg.". The rightmost column is the overall average of all MMDs on two datasets. For baselines, we directly ported the results from You et al. (2018b) and Liu et al. (2019). For a fair comparison, we followed the settings of evaluation in Liu et al. (2019).

Score-Based Graph Generation

Expriments

- We also the power of 1) learnable graph adjacency matrix and 2) multi-channels.

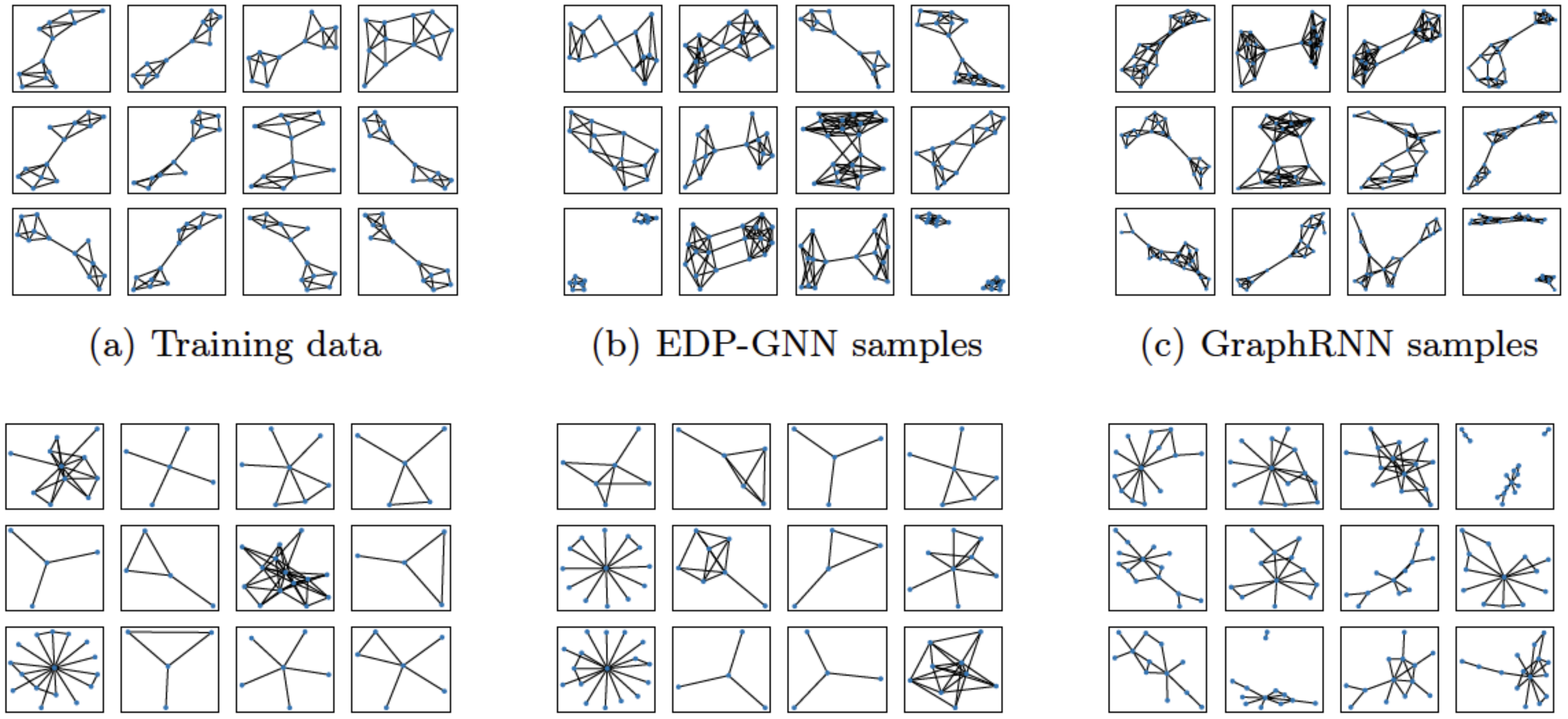


A*	C*	**Community-small**		**Ego-small**	
		Train loss	Test loss	Train loss	Test loss
N	N	140	140	14	17
Y	N	120	120	12	15
N	Y	110	120	13	15
Y	Y	**98**	**96**	**10**	**12**

Table 3: Ablation experiments on Community-small and Ego-small datasets. The training and test losses are defined by (3). A* indicates whether the adjacency matrix is learnable, and C* indicates whether the intermediate adjacency matrices have multi-channels.

$$\mathcal{L}(\boldsymbol{\theta}; \{\sigma_i\}_{i=1}^{L}) \triangleq \frac{1}{2L} \sum_{i=1}^{L} \sigma_i^2 \mathbb{E}\left[\left\| \mathbf{s}_{\boldsymbol{\theta}}(\tilde{\mathbf{A}}, \sigma) + \frac{\tilde{\mathbf{A}} - \mathbf{A}}{\sigma^2} \right\|_2^2 \right]. \quad (3)$$

Score-Based Graph Generation
Expriments



(a) Training data (b) EDP-GNN samples (c) GraphRNN samples

(d) Training data (e) EDP-GNN samples (f) GraphRNN samples

Figure 3: Samples from the training data, EDP-GNN, and GraphRNN, on Community-small (top row) and Ego-small (bottom row).

2. DF for Graphs with Discrete State Spaces

DF for Graphs with Discrete State Spaces

Problem Setting

- DDPM is a parameterized Markov Chain trained to reverse a iterative forward process, that gradually transforms a sample into pure noises.

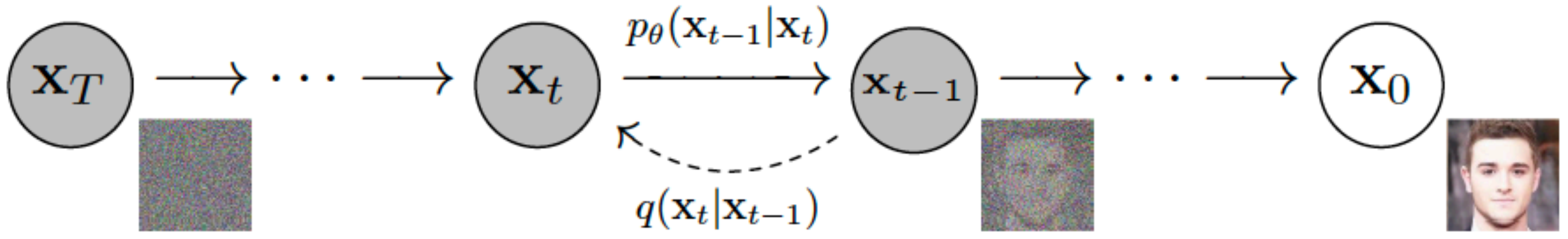

$\mathbf{x}_T$
$\mathbf{x}_t$
$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$
$q(\mathbf{x}_t|\mathbf{x}_{t-1})$
$\mathbf{x}_{t-1}$
$\mathbf{x}_0$

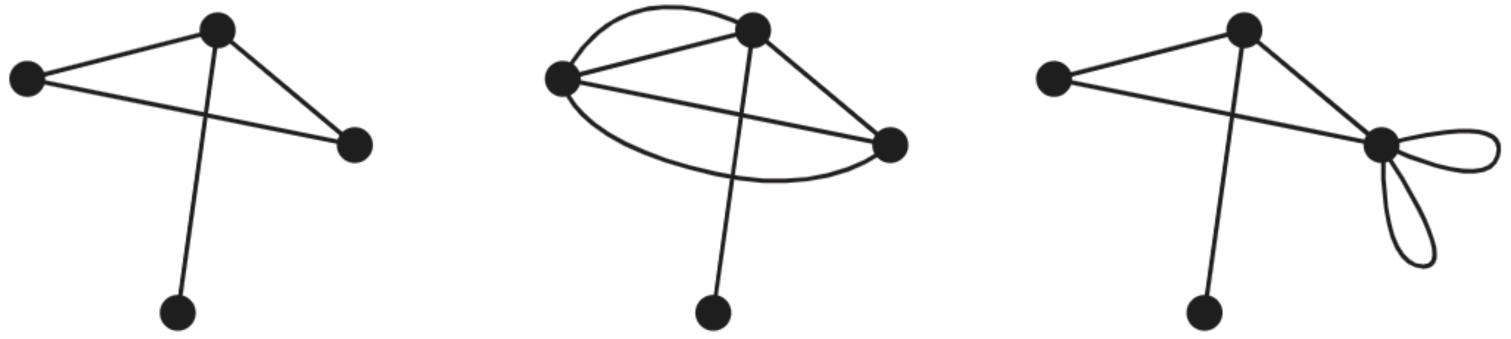
Figure 2: The directed graphical model considered in this work.

- Previous works focus on Gaussian diffusion processes. This contribution suggests adapting to discrete noise and an graph distribution, leading to a random graph model.

- This work focus on the generation of simple graphs, by **modelling the prabability of occurrence of an edge.**

DF for Graphs with Discrete State Spaces
Problem Setting

- Definition of simple graphs



simple graph

nonsimple graph with multiple edges

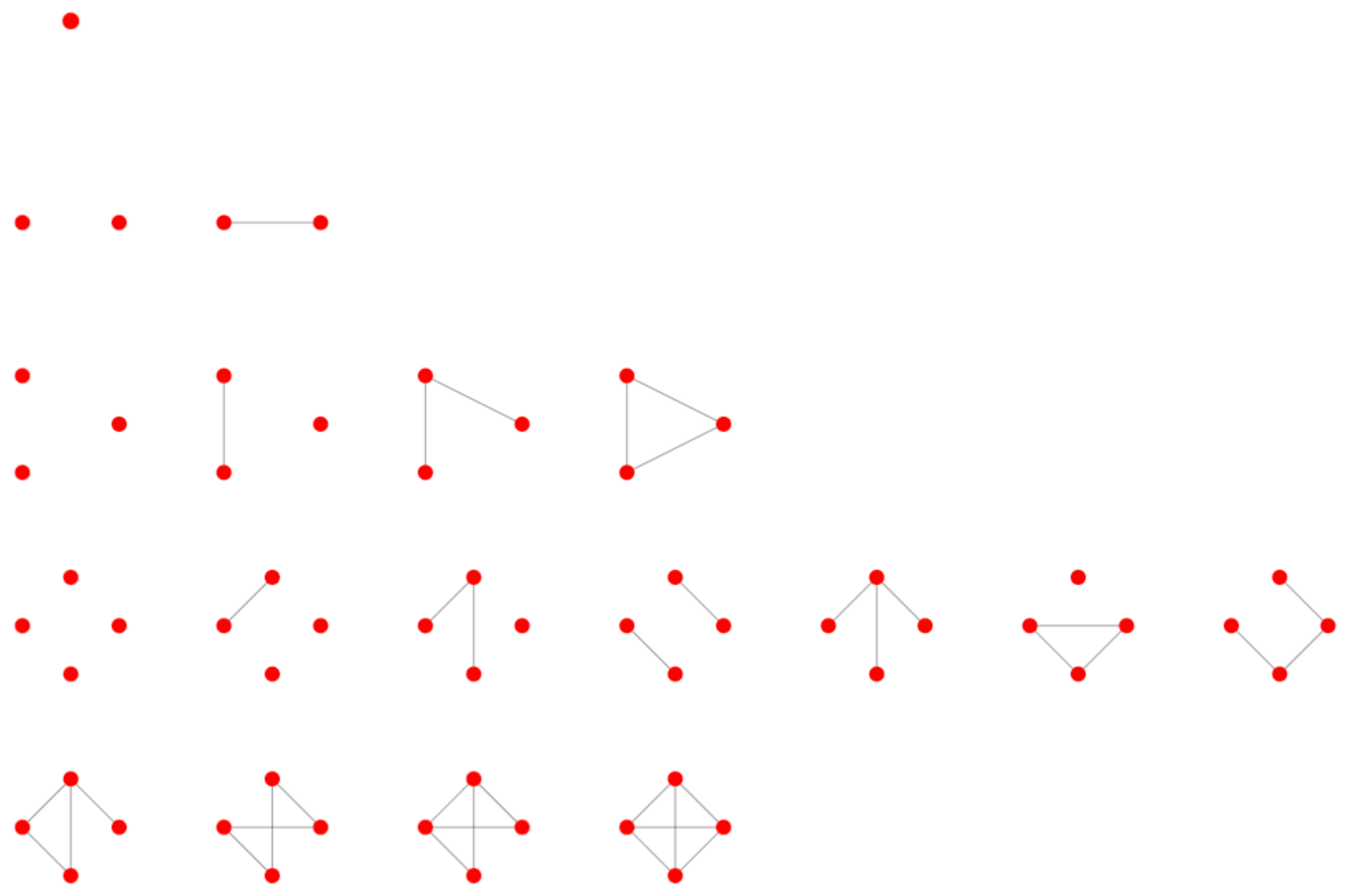
nonsimple graph with loops

A simple graph, also called a strict graph (Tutte 1998, p. 2), is an unweighted, undirected graph containing no graph loops or multiple edges (Gibbons 1985, p. 2; West 2000, p. 2; Bronshtein and Semendyayev 2004, p. 346). A simple graph may be either connected or disconnected.

DF for Graphs with Discrete State Spaces

Problem Setting

- Examples of simple graphs



DF for Graphs with Discrete State Spaces
Forward Process

- Let A_t be a simple graph at time t. The forward process

  $$q(\boldsymbol{A}_{1:T} \mid \boldsymbol{A}_0) = \prod_{t=1}^{T} q(\boldsymbol{A}_t \mid \boldsymbol{A}_{t-1})$$

  generates a sequences of increasingly noisier latent variables A_t from the initial sample A_0.

- Here the sample A_0 and latent variables A_t are represented by adjacency matrices. For each (i, j) element of the adjacency matrix A_t, there is an associating row vector

  $$\boldsymbol{a}_t^{ij} \in \{0, 1\}^2$$

  describing the probability of the occurrence of this edge.

DF for Graphs with Discrete State Spaces

Forward Process

- The forward process is described by multiplication with a **double stochastic matrix** Q_t

  $$\boldsymbol{a}_t^{ij} = \boldsymbol{a}_{t-1}^{ij}\boldsymbol{Q}_t$$

  Here the matrix $Q_t \in \mathbb{R}^{2\times2}$ is modelled as

  $$\boldsymbol{Q}_t = \begin{bmatrix} 1-\beta_t & \beta_t \\ \beta_t & 1-\beta_t \end{bmatrix},$$

  where β_t is interpreted as the probability of changing the edge state.

- This formulation allows direct sampling at any timestep (like the gaussian forward process). Indeed the matrix $\overline{\boldsymbol{Q}}_t = \prod_{i<t}\boldsymbol{Q}_i$ is expressed in the form of

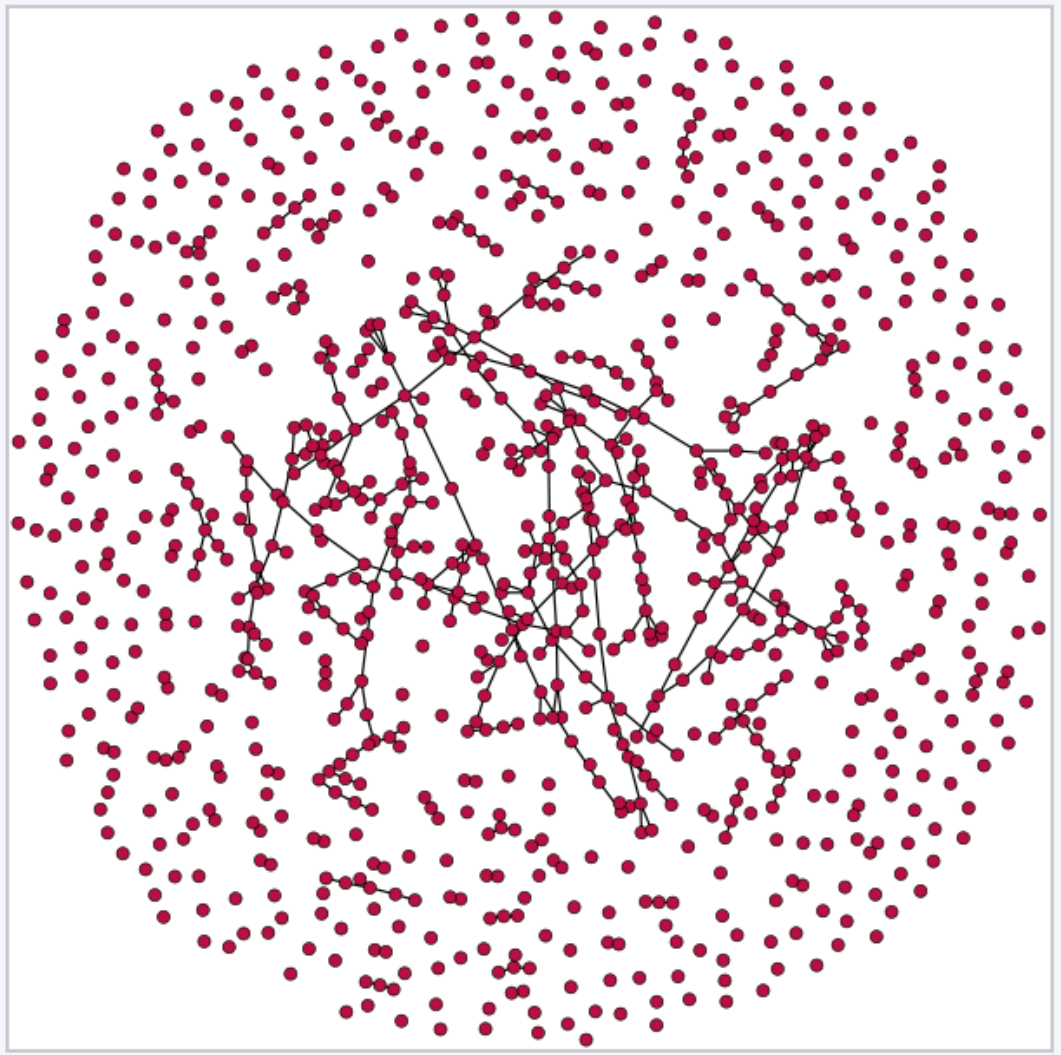
  $$\overline{\beta}_t = \frac{1}{2} - \frac{1}{2}\prod_{i<t}(1-2\beta_i).$$

- Eventually, we want the probability $\bar{\beta} \in [0,0.5]$ to vary from 0 (unperturbed sample) to 0.5 (pure noise).

DF for Graphs with Discrete State Spaces

Forward Process

- The ‘pure noise’ is Erdos–Rényi random $G(n,0.5)$ graphs.
- In the $G(n, M)$ model, a graph is chosen uniformly at random from the collection of all graphs which have n nodes and M edges.
- In the $G(n, p)$ model, a graph is constructed by connecting labeled nodes randomly with probability p.


An Erdős–Rényi–Gilbert graph with 1000 vertices at the critical edge probability $p = 1/(n-1)$, showing a large component and many small ones

DF for Graphs with Discrete State Spaces

Reverse Process

- Similar to DDPM, we construct the reverse process by evalutating

$$q(\boldsymbol{A}_{t-1}|\boldsymbol{A}_t, \boldsymbol{A}_0) = q(\boldsymbol{A}_t|\boldsymbol{A}_{t-1})\frac{q(\boldsymbol{A}_{t-1}|\boldsymbol{A}_0)}{q(\boldsymbol{A}_t|\boldsymbol{A}_0).}$$

which gives (for the (i, j) element of A)

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

- The remaining part the to predict A_0 from A_t.

DF for Graphs with Discrete State Spaces
Loss function

- The upper bound of negative log marginal likelihood is

$$-\log\left(p_\theta\left(\boldsymbol{A}_0\right)\right) \leq \mathbb{E}_{\boldsymbol{A}_{1:T}\sim q(\boldsymbol{A}_{1:T}|\boldsymbol{A}_0)}\left[\log\left(\frac{p_\theta\left(\boldsymbol{A}_{0:T}\right)}{q\left(\boldsymbol{A}_{1:T} \mid \boldsymbol{A}_0\right)}\right)\right] := L_{\text{vb}}(\boldsymbol{A}_0)$$

$$\begin{aligned}
L_{\text{vb}} &= \mathbb{E}_{q(\boldsymbol{A}_{0:T})}\left[-\log\left(\frac{p_\theta\left(\boldsymbol{A}_{0:T}\right)}{q\left(\boldsymbol{A}_{1:T} \mid \boldsymbol{A}_0\right)}\right)\right] \\
&= \mathbb{E}_q\left[-\log\left(p_\theta\left(\boldsymbol{A}_T\right)\right) - \sum_{t=1}^{T}\log\left(\frac{p_\theta\left(\boldsymbol{A}_{t-1} \mid \boldsymbol{A}_t\right)}{q\left(\boldsymbol{A}_t \mid \boldsymbol{A}_{t-1}\right)}\right)\right] \\
&= \mathbb{E}_q\left[-\log\left(p_\theta\left(\boldsymbol{A}_T\right)\right) - \sum_{t=2}^{T}\log\left(\frac{p_\theta\left(\boldsymbol{A}_{t-1} \mid \boldsymbol{A}_t\right)}{q\left(\boldsymbol{A}_t \mid \boldsymbol{A}_{t-1}\right)}\right) - \log\left(\frac{p_\theta\left(\boldsymbol{A}_0 \mid \boldsymbol{A}_1\right)}{q\left(\boldsymbol{A}_1 \mid \boldsymbol{A}_0\right)}\right)\right] \\
&= \mathbb{E}_q\left[-\log\left(p_\theta\left(\boldsymbol{A}_T\right)\right) - \sum_{t=2}^{T}\log\left(\frac{p_\theta\left(\boldsymbol{A}_{t-1} \mid \boldsymbol{A}_t\right)}{q\left(\boldsymbol{A}_{t-1} \mid \boldsymbol{A}_t, \boldsymbol{A}_0\right)} \cdot \frac{q\left(\boldsymbol{A}_{t-1} \mid \boldsymbol{A}_0\right)}{q\left(\boldsymbol{A}_t \mid \boldsymbol{A}_0\right)}\right) - \log\left(\frac{p_\theta\left(\boldsymbol{A}_0 \mid \boldsymbol{A}_1\right)}{q\left(\boldsymbol{A}_1 \mid \boldsymbol{A}_0\right)}\right)\right] \quad (6) \\
&= \mathbb{E}_q\left[-\log\left(\frac{p_\theta\left(\boldsymbol{A}_T\right)}{q\left(\boldsymbol{A}_T \mid \boldsymbol{A}_0\right)}\right) - \sum_{t=2}^{T}\log\left(\frac{p_\theta\left(\boldsymbol{A}_{t-1} \mid \boldsymbol{A}_t\right)}{q\left(\boldsymbol{A}_{t-1} \mid \boldsymbol{A}_t, \boldsymbol{A}_0\right)}\right) - \log\left(p_\theta\left(\boldsymbol{A}_0 \mid \boldsymbol{A}_1\right)\right)\right] \\
&= \mathbb{E}_{\mathbb{E}_{q(\boldsymbol{A}_0)}}\Bigg[D_{KL}(q(\boldsymbol{A}_T|\boldsymbol{A}_0)\|p_\theta(\boldsymbol{A}_T)) + \sum_{t=2}^{T}\mathbb{E}_{q(\boldsymbol{A}_t|\boldsymbol{A}_0)}D_{KL}(q(\boldsymbol{A}_{t-1}|\boldsymbol{A}_t,\boldsymbol{A}_0)\|p_\theta(\boldsymbol{A}_{t-1}|\boldsymbol{A}_t)) \\
&\quad -\mathbb{E}_{q(\boldsymbol{A}_1|\boldsymbol{A}_0)}\log(p_\theta(\boldsymbol{A}_0|\boldsymbol{A}_1))\Bigg]
\end{aligned}$$

DF for Graphs with Discrete State Spaces

Loss function

- Finally, we use the loss

$$L_{\text{vb}}(\boldsymbol{A}_0)) := \mathbb{E}_{q(\boldsymbol{A}_0)}\left[\underbrace{D_{KL}(q(\boldsymbol{A}_T|\boldsymbol{A}_0)\|p_\theta(\boldsymbol{A}_T))}_{L_T}\right.$$
$$\left.+\sum_{t=1}^{T}\mathbb{E}_{q(\boldsymbol{A}_t|\boldsymbol{A}_0)}\underbrace{D_{KL}(q(\boldsymbol{A}_{t-1}|\boldsymbol{A}_t,\boldsymbol{A}_0)\|p_\theta(\boldsymbol{A}_{t-1}|\boldsymbol{A}_t))}_{L_t}\underbrace{-\mathbb{E}_{q(\boldsymbol{A}_1|\boldsymbol{A}_0)}\log(p_\theta(\boldsymbol{A}_0|\boldsymbol{A}_1))}_{L_0}\right]$$

- We use a network $NN_\theta(A_t)$ to predict the logits of the distribution $p_\theta(A_0\,|\,A_t)$, then recover $p_\theta(A_{t-1}\,|\,A_t)$ from A_t and A_0.

DF for Graphs with Discrete State Spaces
Loss function

- We may also simplify the KL divergence term L_t.

$$\begin{aligned} D_{KL}\left(q\left(\boldsymbol{A}_{t-1} \mid \boldsymbol{A}_t, \boldsymbol{A}_0\right) \| p_\theta\left(\boldsymbol{A}_{t-1} \mid \boldsymbol{A}_t\right)\right) &= \mathbb{E}_{q\left(\boldsymbol{A}_{t-1} \mid \boldsymbol{A}_t, \boldsymbol{A}_0\right)}\left[-\log\left(\frac{p_\theta\left(\boldsymbol{A}_{t-1} \mid \boldsymbol{A}_t\right)}{q\left(\boldsymbol{A}_{t-1} \mid \boldsymbol{A}_t, \boldsymbol{A}_0\right)}\right)\right] \\ &= \mathbb{E}_{q\left(\boldsymbol{A}_{t-1} \mid \boldsymbol{A}_t, \boldsymbol{A}_0\right)}\left[-\log\left(p_\theta\left(\boldsymbol{A}_0 \mid \boldsymbol{A}_t\right)\right)\right] \\ &= -\log\left(p_\theta\left(\boldsymbol{A}_0 \mid \boldsymbol{A}_t\right)\right) \end{aligned}$$

- This gives a simpler loss function

$$L_{\text{simple}} := -\mathbb{E}_{q\left(\boldsymbol{A}_0\right)} \sum_{t=1}^{T}\left(1 - 2 \cdot \overline{\beta}_t + \frac{1}{T}\right) \cdot \mathbb{E}_{q\left(\boldsymbol{A}_t \mid \boldsymbol{A}_0\right)} \log p_\theta\left(\boldsymbol{A}_0 \mid \boldsymbol{A}_t\right))$$

where we use a linear reweighting machanism proposed in [Sam et al., 2022].

DF for Graphs with Discrete State Spaces

Sampling

- We start by sampling each edge independently from a Bernoulli distribution with probability $p = 1/2$, denoted by $\mathscr{B}_{p=1/2}$.

Algorithm 1 Sampling for L_{vb}

```
∀$i, j | i > j$: $\boldsymbol{A}_T^{ij} \sim \mathcal{B}_{p=1/2}$
for $t = T, ..., 1$ do
    Compute $p_\theta(\boldsymbol{A}_{t-1}|\boldsymbol{A}_t)$
    $\boldsymbol{A}_{t-1} \sim p_\theta(\boldsymbol{A}_{t-1}|\boldsymbol{A}_t)$
end for
```

Algorithm 2 Sampling for L_{simple}

```
∀$i, j | i > j$: $\boldsymbol{A}_T^{ij} \sim \mathcal{B}_{p=1/2}$
for $t = T, ..., 1$ do
    $\tilde{\boldsymbol{A}}_0 \sim p_\theta(\boldsymbol{A}_0|\boldsymbol{A}_t)$
    $\boldsymbol{A}_{t-1} \sim q(\boldsymbol{A}_{t-1}|\tilde{\boldsymbol{A}}_0)$
end for
```

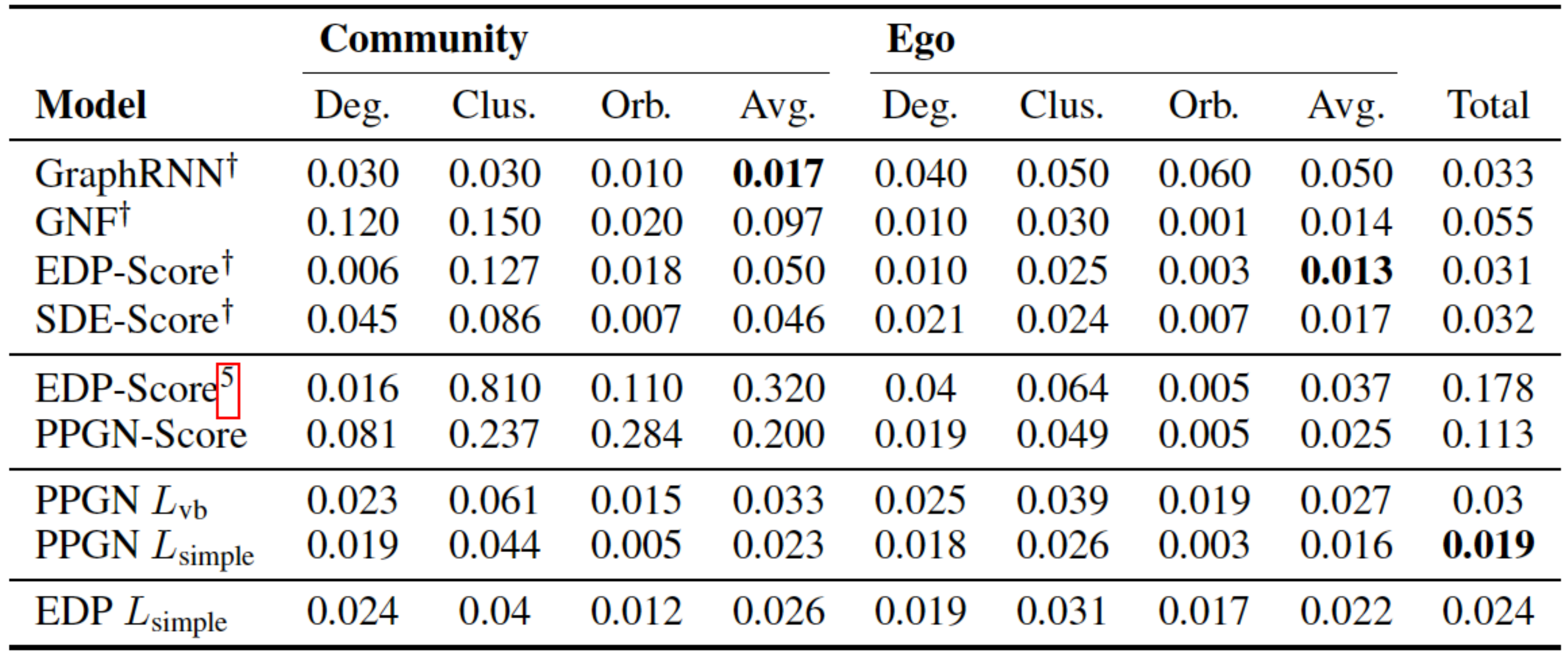
- **Noise schedule**. The values of $\bar{\beta}_t$ are selected following a linear schedule.

DF for Graphs with Discrete State Spaces

Experiments

- We test the proposed model on two datasets, Community-small ($12 \leq N \leq 20$) and Ego-small ($4 \leq N \leq 18$). Our baselines EDP-GNN and PPGN.

- We calculate the MMD of of the distribution of three graph statistics. 1) degree distribution; 2) cluster coefficient distribuition and 3) the number of orbits of 4 nodes.



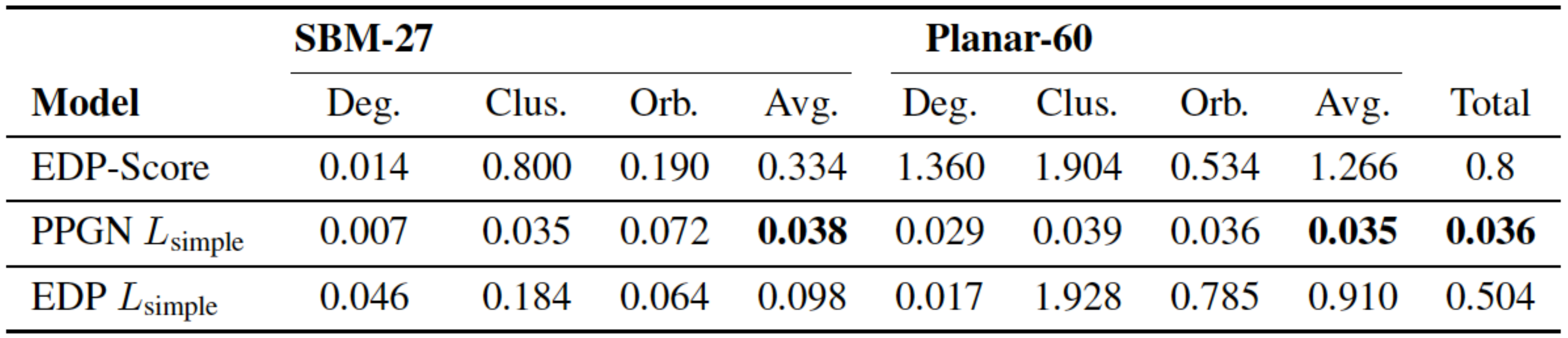
	Community				**Ego**				
Model	Deg.	Clus.	Orb.	Avg.	Deg.	Clus.	Orb.	Avg.	Total
GraphRNN†	0.030	0.030	0.010	**0.017**	0.040	0.050	0.060	0.050	0.033
GNF†	0.120	0.150	0.020	0.097	0.010	0.030	0.001	0.014	0.055
EDP-Score†	0.006	0.127	0.018	0.050	0.010	0.025	0.003	**0.013**	0.031
SDE-Score†	0.045	0.086	0.007	0.046	0.021	0.024	0.007	0.017	0.032
EDP-Score[5]	0.016	0.810	0.110	0.320	0.04	0.064	0.005	0.037	0.178
PPGN-Score	0.081	0.237	0.284	0.200	0.019	0.049	0.005	0.025	0.113
PPGN L_{vb}	0.023	0.061	0.015	0.033	0.025	0.039	0.019	0.027	0.03
PPGN L_{simple}	0.019	0.044	0.005	0.023	0.018	0.026	0.003	0.016	**0.019**
EDP L_{simple}	0.024	0.04	0.012	0.026	0.019	0.031	0.017	0.022	0.024

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.

DF for Graphs with Discrete State Spaces

Experiments

- We also compare the results on two more challenging dataset, stochastic-block-model ($24 \leq n \leq 27$) and a planar dataset with $n = 60$ nodes.



	SBM-27				**Planar-60**				
Model	Deg.	Clus.	Orb.	Avg.	Deg.	Clus.	Orb.	Avg.	Total
EDP-Score	0.014	0.800	0.190	0.334	1.360	1.904	0.534	1.266	0.8
PPGN L_{simple}	0.007	0.035	0.072	**0.038**	0.029	0.039	0.036	**0.035**	**0.036**
EDP L_{simple}	0.046	0.184	0.064	0.098	0.017	1.928	0.785	0.910	0.504

Table 2: MMD results for the SBM-27 and the Planar-60 datasets. The "Total" column denotes the average MMD over all of the 6 measurements.

- The authors comments that they use 32 timesteps, while EDP-Score use 1000. The proposed discrete formulation is even more beneficial when graph size and complexity increase.

DF for Graphs with Discrete State Spaces

Experiments

- EDP-Score fails to generate graphs of high complexity.

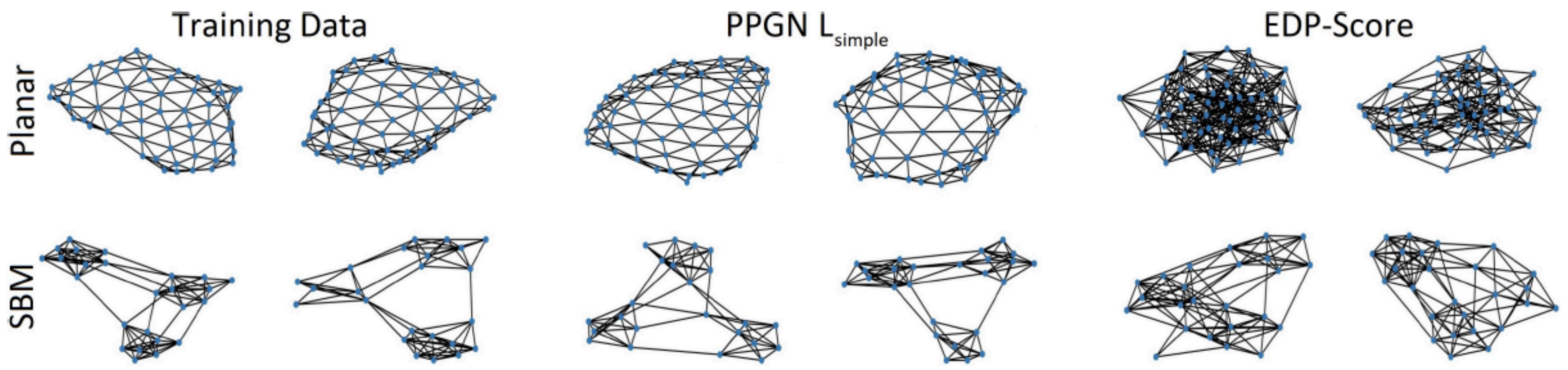


Figure 1: Sample Planar-60 and SBM-27 graphs generated by our approach (PPGN L_{simple} - Middle) and the original Gaussian Score-matching approach (EDP-Score - Right).