Permutation Invariant Graph Generation via Score-Based Generative Modeling

Chenhao Niu[1], **Yang Song**[2], **Jiaming Song**[2], **Shengjia Zhao**[2], **Aditya Grover**[2], **Stefano Ermon**[2]
[1]Tsinghua University [2]Stanford University

Abstract

Learning generative models for graph-structured data is challenging because graphs are discrete, combinatorial, and the underlying data distribution is invariant to the ordering of nodes. However, most of the existing generative models for graphs are not invariant to the chosen ordering, which might lead to an undesirable bias in the learned distribution. To address this difficulty, we propose a permutation invariant approach to modeling graphs, using the recent framework of score-based generative modeling. In particular, we design a permutation equivariant, multi-channel graph neural network to model the gradient of the data distribution at the input graph (a.k.a., the score function). This permutation equivariant model of gradients implicitly defines a permutation invariant distribution for graphs. We train this graph neural network with score matching and sample from it with annealed Langevin dynamics. In our experiments, we first demonstrate the capacity of this new architecture in learning discrete graph algorithms. For graph generation, we find that our learning approach achieves better or comparable results to existing models on benchmark datasets.

1 INTRODUCTION

Graphs are used to capture relational structure in many domains, including knowledge bases (Hamaguchi et al., 2017), social networks (Hamilton et al., 2017; Kipf and Welling, 2016), protein interaction networks (Fout et al., 2017), and physical systems (Batagelj and Zaversnik, 2003). Generating graphs using suitable probabilistic models has many applications, such as drug design (Duvenaud et al., 2015; Gómez-Bombarelli et al., 2018; Li et al., 2018a), creating computation graphs for architecture search (Xie et al., 2019), as well as research in network science (Watts and Strogatz, 1998; Albert and Barabási, 2002; Leskovec et al., 2010).

While many stochastic models of graphs have been proposed, the idea of learning statistical generative models of graphs from data has recently gained significant attention. One approach is to use latent variable generative models similar to variational autoencoders (Kingma and Welling, 2013). Examples include GraphVAE (Simonovsky and Komodakis, 2018), Graphite (Grover et al., 2018), and junction tree variational autoencoders (Jin et al., 2018). These models typically use a graph neural network (GNN) (Gori et al., 2005; Scarselli et al., 2008) to encode graph data to a latent space, and generate samples by decoding latent variables sampled from a prior distribution. The second paradigm is autoregressive graph generative models (Li et al., 2018a; You et al., 2018a; Liao et al., 2019), where graphs are generated sequentially, one node (or one subgraph) at a time.

Although these models have achieved great success, they are not satisfying in terms of capturing the permutation invariance properties of graphs. Permutation invariance is a fundamental inductive bias of graph-structured data. For a graph with N nodes, there are up to $N!$ different adjacency matrices that are equivalent representations of the same graph. Therefore, a graph generative model should ideally assign the same probability to each of these equivalent adjacency matrices. It is challenging, however, to enforce permutation invariance in variational autoencoders or autoregressive models. Some previous approaches only approximately induce permutation invariance: GraphVAE (Simonovsky and Komodakis, 2018) uses inexact graph matching techniques requiring up to $O(N^4)$ operations, whereas the model in Li et al. (2018a) augments the training data by randomly permuting the nodes of existing data. Other approaches instead

Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS) 2020, Palermo, Italy. PMLR: Volume 108.

focus on selecting a specific node ordering based on heuristics: GraphRNN (You et al., 2018b) uses random breadth-first search (BFS) to determine an ordering, and GRAN (Liao et al., 2019) adaptively chooses an ordering depending on the input graph from a family of pre-defined node orderings.

To better capture the permutation invariance of graphs, we propose a new graph generative model using the framework of score-based generative modeling (Song and Ermon, 2019). Intuitively, this approach trains a model to capture the vector field of gradients of the log data density of graphs (a.k.a., scores). Contrary to likelihood-based models such as variational auto-encoders and autoregressive models, score-based generative modeling imposes fewer constraints on the model architectures (e.g., a score does not have to be normalized). This enables the use of function families with desirable inductive biases, such as permutation invariance. In particular, we leverage graph neural networks (Scarselli et al., 2008) to build a permutation equivariant model for the scores of the distribution over graphs we wish to learn. As shown later in the paper, this implicitly defines a permutation invariant distribution over adjacency matrices representing graphs.

As in other classes of deep generative models, the neural architecture used in score-based generative modeling is critical to its success. In this work, we introduce a new type of graph neural networks, named EDP-GNN, with learnable multi-channel adjacency matrices. In our experiments, we first test the effectiveness of EDP-GNN for the task of learning graph algorithms, where it significantly outperforms traditional GNNs. Next, we evaluate the generation quality of our score-based models using MMD (Gretton et al., 2012) metrics on several graph datasets, where we achieved comparable performance to GraphRNN (You et al., 2018b), a competitive method for generative modeling of graphs.

2 PRELIMINARIES

2.1 Notations

For each weighted undirected graph, we can choose an ordering of nodes π and represent it with an adjacency matrix $\mathbf{A}^\pi$. Here we use the superscript π to indicate that the rows/columns of $\mathbf{A}^\pi$ are arranged in accordance with a specific node ordering π. When the graph is undirected, the corresponding adjacency matrix $\mathbf{A}^\pi$ is symmetric. We denote the set of adjacency matrices as $\mathcal{A} = \{\mathbf{A} \in \mathbb{R}^{N\times N} \mid \mathbf{A} = \mathbf{A}^\intercal, N \in \mathbb{N}^+\}$.

A distribution of graphs can be represented as a distribution of adjacency matrices $p(\mathbf{A}^\pi)$. Since graphs are invariant to permutations, $\mathbf{A}^{\pi_1}$ and $\mathbf{A}^{\pi_2}$ always represent the same graph for any different node orderings π_1 and π_2. This permutation invariance also implies that $\forall \pi_1 \neq \pi_2 : p(\mathbf{A}^{\pi_1}) = p(\mathbf{A}^{\pi_2})$, *i.e.*, the distribution of adjacency matrices is invariant to node permutations. In the sequel, we often omit the superscript π in $\mathbf{A}^\pi$ when not emphasizing any specific node ordering.

2.2 Graph Neural Network (GNN)

Graph neural networks are a family of neural networks that map graphs to vector representations using message-passing type operations on node features (Gori et al., 2005; Scarselli et al., 2008). They are natural models for graph-structured data; for example, GIN (Xu et al., 2018a) is one type of GNN that is proved to be as expressive as the Weisfeiler-Lehman graph isomorphism test (WL-test). The message passing mechanism guarantees that the output representation $f_{\text{GNN}}(\mathbf{A}^\pi)$ of an input adjacency matrix $\mathbf{A}^\pi$ is *equivariant to permutations* of the node ordering π.

2.3 Score-Based Generative Modeling

Score-based generative modeling (Song and Ermon, 2019) is a class of generative models. For a probability density function $p(\mathbf{x})$, the score function is defined as $\nabla_{\mathbf{x}} \log p(\mathbf{x})$. Instead of directly modeling the density function of the data distribution $p_{\text{data}}(\mathbf{x})$, score-based generative modeling estimates the data score function $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$. The advantage is that *the score function can be easier to model than the density function.*

For better score estimation, following (Song and Ermon, 2019) we perturb the data with Gaussian noise of different intensities, and estimate the scores jointly for all noise levels. We train a noise conditional model $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}; \sigma)$ (e.g., a neural network parameterized by $\boldsymbol{\theta}$) to approximate the score function corresponding to noise level σ. Given a data distribution $p_{\text{data}}(\mathbf{x})$, a noise distribution $q_\sigma(\tilde{x} \mid \mathbf{x})$ (*e.g.*, $\mathcal{N}(\tilde{\mathbf{x}} \mid \mathbf{x}, \sigma^2 I)$), and a sequence of noise levels $\{\sigma_i\}_{i=1}^L$, the training loss $\mathcal{L}(\boldsymbol{\theta}; \{\sigma_i\}_{i=1}^L)$ is defined as:

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

where the expectation is taken with respect to the sampling process: $\mathbf{x} \sim p_{\text{data}}(\mathbf{x}), \tilde{\mathbf{x}} \sim q_{\sigma_i}(\tilde{\mathbf{x}} \mid \mathbf{x})$. We note that all expectations in $\mathcal{L}(\boldsymbol{\theta}; \{\sigma_i\}_{i=1}^L)$ can be estimated with i.i.d. samples from $p_{\text{data}}(\mathbf{x})$ and $q_\sigma(\cdot|\mathbf{x})$, which are easy to obtain. The objective is $\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; \{\sigma_i\}_{i=1}^L)$.

After the conditional score model $s_{\boldsymbol{\theta}}(\mathbf{x}; \sigma)$ has been trained, we use annealed Langevin dynamics (Song and Ermon, 2019) for sample generation (see Algorithm 1).

Algorithm 1 Annealed Langevin dynamics sampling.

Require: $\{\sigma_i\}_{i=1}^{L}, \epsilon, T$ ▷ ϵ is smallest step size; T is the number of iteration for each noise level.

Initialize $\tilde{\mathbf{x}}_0$
for $i \leftarrow 1$ to L **do**
  $\alpha_i \leftarrow \epsilon \cdot \sigma_i^2/\sigma_L^2$ ▷ α_i is the step size.
  for $t \leftarrow 1$ to T **do**
    Draw $\mathbf{z}_t \sim \mathcal{N}(0, I)$
    $\tilde{\mathbf{x}}_t \leftarrow \tilde{\mathbf{x}}_{t-1} + \frac{\alpha_i}{2}\mathbf{s}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}_{t-1}, \sigma_i) + \sqrt{\alpha_i}\ \mathbf{z}_t$
  end for
  $\tilde{\mathbf{x}}_0 \leftarrow \tilde{\mathbf{x}}_T$
end for
return $\tilde{\mathbf{x}}_T$

3 SCORE-BASED GENERATIVE MODELING FOR GRAPHS

Contrary to the weighted graphs we used to define the probability density function in Section 2.1, in real-world problems unweighted graphs are much more common, which means entries in the adjacency matrix $\mathbf{A}$ can only be either 0 or 1. While the score-based method (Song and Ermon, 2019) was initially proposed for handling continuous data, it can be adopted to generate discrete ones as well. Below, we first show our modifications of score-based generative modeling for graph generation, and then introduce our specialized neural network architecture EDP-GNN for the noise conditional model $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{A}; \sigma)$, where $\mathbf{s}_{\boldsymbol{\theta}}(\cdot; \sigma) : \mathcal{A} \to \mathcal{A}$.

3.1 Noise Distribution

We add Gaussian perturbations to adjacency matrices and define the noise distribution $q_\sigma(\tilde{\mathbf{A}} \mid \mathbf{A})$ as follows

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

Intuitively, we only add Gaussian noise to the upper triangular part of the adjacency matrix, because we focus on undirected graphs whose adjacency matrices are symmetric.

Since $\nabla_{\tilde{\mathbf{A}}} \log q_\sigma(\tilde{\mathbf{A}}|\mathbf{A}) = -(\tilde{\mathbf{A}} - \mathbf{A})/\sigma^2$, the training loss of $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{A}, \sigma)$ is

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

where the expectation is over the sampling process defined via $\mathbf{A} \sim p_{\text{data}}(\mathbf{A})$ and $\tilde{\mathbf{A}} \sim q_\sigma(\tilde{\mathbf{A}}|\mathbf{A})$. The objective is $\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; \{\sigma_i\}_{i=1}^{L})$.

Note that the supports of the noise distributions $\{q_{\sigma_i}\}_{i=1}^{L}$ span $\mathbb{R}^{N \times N}$, where N is the number of nodes of the input graph. Therefore, the scores of perturbed distributions corresponding to all noise levels are well-defined, regardless of whether the training samples are discrete or not.

3.2 Sampling

To generate $\tilde{\mathbf{A}}$, we first sample N, which is the number of nodes to be generated, and then sample $\tilde{\mathbf{A}} \in \mathbb{R}^{N \times N}$ with annealed Langevin dynamics. This amounts to factorizing $p(\mathbf{A}) = \sum_{N=1}^{\infty} p(\mathbf{A} \mid \mathbf{A} \in \mathbb{R}^{N \times N}) p(N)$. Implementation-wise, we sample N from the empirical distribution of number of nodes in the training dataset, as done in (Li et al., 2018b). When doing annealed Langevin dynamics, we first initialize $\tilde{\mathbf{A}}_0$ 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 $\varepsilon_{[i,j]} \sim \mathcal{N}(0, 1)$. Then, we update $\tilde{\mathbf{A}}$ by iteratively sampling from a series of trained conditional score models $\{\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{A}; \sigma_i)\}_{i=1}^{L}$ using Langevin dynamics. For each of the conditional score model $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{A}; \sigma_i)$, we run Langevin dynamics for T steps, where the series $\{\sigma_i\}_{i=1}^{L}$ is annealed down over the process such that σ_1 is large but σ_L is small enough that it can be ignored. As a minor modification, we change the noise term $\mathbf{z}_t$ in Algorithm 1 to a symmetric one $\tilde{\mathbf{z}}_t$, given by

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

which accouts for the symmetry of adjacency matrices.

Score-based generative modeling provides samples in the continuous space, whereas graph data are often discrete. In order to obtain discrete samples, we quantize the generated continuous adjacency matrix (denoted as $\tilde{\mathbf{A}}$) to a binary one (denoted as $\mathbf{A}^{(\text{sample})}$) at the end of annealed Langevin dynamics. Formally, this quantization operation is defined as

$$\mathbf{A}_{[i,j]}^{(\text{sample})} = \mathbb{1}_{\tilde{\mathbf{A}}_{[i,j]} > 0.5} \tag{4}$$

where $\mathbb{1}$ is an indicator function that evalutes to 1 when the condition holds and 0 otherwise.

3.3 Permutation Equivariance and Invariance

Permutation invariance is a desirable property of graph generative models, since the true distribution $p_{\text{data}}(\mathbf{A})$ is inherently permutation invariant. We show that by using a permutation equivariant score function $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{A}; \sigma)$, the corresponding distribution is permutation invarant.

Theorem 1. *If* $\mathbf{s}: \mathbb{R}^{N\times N} \to \mathbb{R}^{N\times N}$ *is a permutation equivariant function, then the scalar function* $f_{\mathbf{s}} = \int_{\gamma[\mathbf{0},\mathbf{A}]} \langle \mathbf{s}(\mathbf{X}), \mathrm{d}\,\mathbf{X}\rangle_{\mathrm{F}} + C$ *is permutation invariant, where* $\langle \mathbf{A}, \mathbf{B}\rangle_{\mathrm{F}} = \mathrm{tr}(\mathbf{A}^\intercal \mathbf{B})$ *is the Frobenius inner product,* $\gamma[\mathbf{0}, \mathbf{A}]$ *is any curve from* $\mathbf{0} = \{0\}_{N\times N}$ *to* $\mathbf{A}$*, and* $C \in \mathbb{R}$ *is a constant.*

Proof. See Appendix B. □

Since the gradient of log-likelihood estimation $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{A}) = \nabla_{\mathbf{A}} \log p_{\boldsymbol{\theta}}(\mathbf{A})$ is permutation equivariant, the implicitly defined log-likelihood function $\log p_{\boldsymbol{\theta}}(\mathbf{A})$ is permutation invariant, according to Theorem 1, given below by the line integral of $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{X})$.

$$\log p_{\boldsymbol{\theta}}(\mathbf{A}) = \int_{\gamma[\mathbf{0},\mathbf{A}]} \langle \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{X}), \mathrm{d}\,\mathbf{X}\rangle_{\mathrm{F}} + \log p_{\boldsymbol{\theta}}(\mathbf{0})$$

3.4 Edgewise Dense Prediction Graph Neural Network (EDP-GNN)

Below, we introduce a GNN-based score network $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{A}; \sigma)$ that can effectively model the scores of graph distributions while being permutation equivariant.

3.4.1 Multi-Channel GNN Layer

We introduce the multi-Channel GNN layer , an extended version of the GIN (Xu et al., 2018a) layer, which serves as a basic component of our EDP-GNN model. The intuition is to run message-passing simultaneously on many different graphs, and collect the node features from all the channels via concatenation. For a C-channel GNN layer with M message-passing steps, the m-th message-passing step can be expressed as follows,

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

where i is the index of nodes, C is the number of channels, $\mathbf{A}^{(k)} \in \mathbb{R}^{C\times N\times N}$ is the multi-channel adjacency matrix, and $\mathbf{Z}^{(m)} \in \mathbb{R}^{N\times F^{(m)}}$ is the vector of node features. Here ϵ is a learnable parameter, the same as in the original GIN, CONCAT stands for the concatenation operation, and $\mathrm{MLP}^{(m)}_{\mathrm{Node}}$ transforms each node feature using a multilayer perceptron.

After M steps of message-passing, we use the same concatenation operation as GIN to obtain node features. Specifically, for each node v_i, the output feature is given by

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

Henceforth, we denote our Multi-Channel GNN layer as

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

3.4.2 EDP-GNN Layer

The EDP-GNN layer is the key component of our model. It transforms the input adjacency matrix to another one, allowing us to adaptively change the process of message passing. The intuition is similar to neural networks for image dense prediction tasks (*e.g.*, semantic parsing), where convolutional layers transform the input image to a feature map in a pixelwise manner, leveraging local information around each pixel location. Similarly, 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.

One EDP-GNN layer has two steps:

1. **Node feature inference:** Using MultiChannelGNN to encode the local structure of different channels of the graph into node features, given by

$$\mathbf{Z}^{(k+1)} = \mathrm{MultiChannelGNN}^{(k)}(\mathbf{A}^{(k)}, \mathbf{Z}^{(k)}); \quad (5)$$

2. **Edge feature inference:** Updating the feature vector of each edge based on the current features of the edge and the updated feature vector of the two endpoints. For each edge $\{v_i, v_j\}$, this operation is given by

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

where $\mathrm{MLP}^{(k)}_{\mathrm{Edge}}$ denotes a multilayer perceptron applied to edge features. To ensure symmetry, the new adjacency matrix is given by

$$\mathbf{A}^{(k+1)} = \tilde{\mathbf{A}}^{(k+1)} + (\tilde{\mathbf{A}}^{(k+1)})^\intercal. \quad (6)$$

3.4.3 Input and Output Layers

Input layer: Input graphs need to be preprocessed before they can be fed into our EDP-GNN model. In particular, we take adjacency matrices of two channels as the input, where the first channel is the original adjacency matrix of an input graph, and the other channel is the negated version of the same adjacency matrix, where each entry is flipped. The node features are initialized using the weighted degrees. Formally,

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

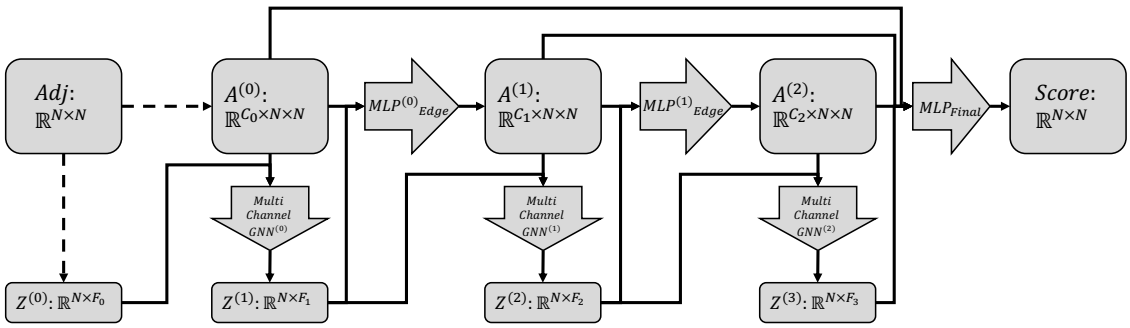


Figure 1: This figure shows an EDP-GNN with three layers. The input is an adjacency matrix of a graph with N nodes given a fixed node ordering, and the outputs are edge representations. The dashed lines are preprocessing steps, and solid lines represent network computations.

where **Adj** is the adjacency matrix of an input graph. If we have node features $\mathbf{X} \in \mathbb{R}^{N\times F_0}$ from data, then we use the following initialization for each node v_i

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

Output layer: To get the output, we employ a similar approach to Xu et al. (2018b), where we aggregate the information from all previous layers to produce a set of permutation equivariant edge features. This can effectively collect information extracted in shallower layers. Formally, for each edge $\{v_i, v_j\}$, the output features are given by

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

3.4.4 Noise Level Conditioning

The framework of score-based generative modeling proposed in (Song and Ermon, 2019) requires a score network conditioned on a series of noise levels. We hope to provide the conditioning on noise levels with as few extra parameters as possible. To this end, we add gains and bias terms conditioned on the index i of the noise level σ_i in all MLP layers, and share all the parameters across different noise levels. A conditional MLP layer for $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{A}, \sigma_i)$ is denoted as

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

where $\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i$ are learnable parameters for each noise level σ_i and activate$(\cdot)$ denotes the activation function. We empirically found that this implementation of noise conditioning achieves similar performance to separately training a score network for each noise level.

3.4.5 Permutation Equivariance of EDP-GNN

The message passing operations in a graph neural network are guaranteed to be permutation equivariant (Keriven and Peyré, 2019), as well as edgewise and nodewise operations for graphs. Since operations in EDP-GNN are either message passing or edgewise/nodewise transformations, the edge features produced by EDP-GNN are guaranteed to be permutation equivariant. In the last EDP-GNN layer, each edge feature is one component of the estimated score. Hence Theorem 1 applies to this score network.

4 RELATED WORK

Flow-Based Graph Generative Models In addition to models mentioned in Section 1, there is also an emerging class of graph generative models based on invertible mappings, such as GNF (Liu et al., 2019) and GraphNVP (Madhawa et al., 2019). These models modify the architecture of a graph neural network (GNN) using coupling layers (Dinh et al., 2016) to enable maximum likelihood learning via the change of variables formula. Since GNNs are permutation invariant, both GNF and GraphNVP could be permutation invariant in principle. However, GraphNVP opts not to be permutation invariant because making their model fully permutation invariant hurts the empirical performance. In contrast, GNF is a permutation invariant model. It achieves permutation invariance by first using a permutation equivariant auto-encoder to encode the graph structure into a set of node features, and then model the distribution of the node features using reversible graph neural networks.

GNNs that Learn Edge Features Although the majority of GNNs focus on node feature learning, (*e.g.*, node classification tasks), there are GNNs, prior to our EDP-GNN, that have intermediate edge features as well. For example, Graph Attention Networks (Veličković et al., 2017) compute an attention coefficient for each edge during message passing (MP) steps. Gong and Cheng (2019) further explored methods to utilize edge

features during the MP steps, such as using normalized attention coefficients to construct a new adjacency matrix for the next MP step, and passing the message simultaneously on multi-input adjacency matrices. However, the model in Gong and Cheng (2019) is not designed for predicting edge features, and the capability to make edgewise prediction is limited by the normalizing operation and the restrictive form of attentions. Kipf et al. (2018) proposed a GNN-based VAE model for relational inference for interacting systems. Contrary to their model which predicts edge information based on only node features, our model takes a weighted graph without node features.

5 EXPERIMENTS

5.1 Learning Graph Algorithms

In this section, we empirically demonstrate the power of the proposed EDP-GNN model on edgewise prediction tasks. In particular, we reduce several classic graph algorithms to the task of predicting whether each edge is in the solution set or not. The training data include a graph and the corresponding solution set, and we train our models to fit the solution set by minimizing the cross-entropy loss.

Setup To verify the ability of EDP-GNN of making edgewise dense predictions, we tested EDP-GNN on learning classic graph algorithms, by labeling all the edges in a graph to indicate whether an edge is in the solution set or not. We choose two simple tasks, 1) Shortest Path (SP) between a given pair of nodes, and 2) Maximum Spanning Tree (MST) of a given graph. The solution set of SP corresponds to a path connecting the pair of nodes with the shortest length, while the solution set of MST is the collection of all edges inside the maximum spanning tree. For both tasks, all the graphs are randomly sampled from the Erdős and Rényi model (E-R) (Erdos and Rényi, 1960) with $n = 12$ and $p = 0.3$. For weighted graphs, all the edge weights are uniformly sampled from $[0, 1]$. A prediction is considered correct if and only if all the labels of the graph are correct. We calculate the accuracy over a fixed test set as the metric. For the baseline model, we use vanilla GIN (Xu et al., 2018a).

Training During training, we generate the training data dynamically on the fly and use the cross-entropy loss as the training objective for both tasks.

Results All results are provided in Tab. 1. We observe that EDP-GNN performs similarly to GIN for unweighted graphs, but achieves much better performance when graphs are weighted. This confirms that EDP-GNN is more 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**

Table 1: The test set accuracy of EDP-GNN vs. GIN on learning the shortest path (SP) and maximum spanning tree (MST) algorithms. "UW" and "W" stand for "unweighted" and "weighted" respectively. Since the training set is dynamically generated, the performance on (newly generated) training set and test set has no difference. Note that for unweighted graphs, there can be more than one shortest path for a given pair of nodes, and the accuracy is underestimated as we randomly picked one as the ground truth, in which case an accuracy of 0.6 is pretty non-trivial.

5.2 Graph Generation Task

In this section, we demonstrate that our EDP-GNN is capable of producing high-quality graph samples via score-based generative modeling. To better understand learnable multi-channel adjacency matrices in our model, we visualize the intermediate channels in Figure 2, and perform extensive ablation studies.

Datasets and Baselines We tested our model on two datasets, Community-small ($12 \leq N \leq 20$) and Ego-small ($4 \leq N \leq 18$), which are also used by You et al. (2018b), and Liu et al. (2019). See Appendix A for more details. 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).

Metrics To evaluate generation quality, we used maximum mean discrepancy (MMD) over some graph statistics, as proposed by You et al. (2018b). We calculated MMD for three graph statistics: 1) degree distribution, 2) cluster coefficient distribution, and 3) the number of orbits with 4 nodes.

Results We compare EDP-GNN against baselines and summarize results in Tab. 2. Our model performs comparably to GraphRNN and GNF with respect to most MMD metrics, and outperforms all other methods when considering the overall average of MMDs on two datasets.

5.2.1 Understanding Intermediate Channels

Intuitively, the intermediate channels of EDP-GNN should be analogous to those in convolutional neural

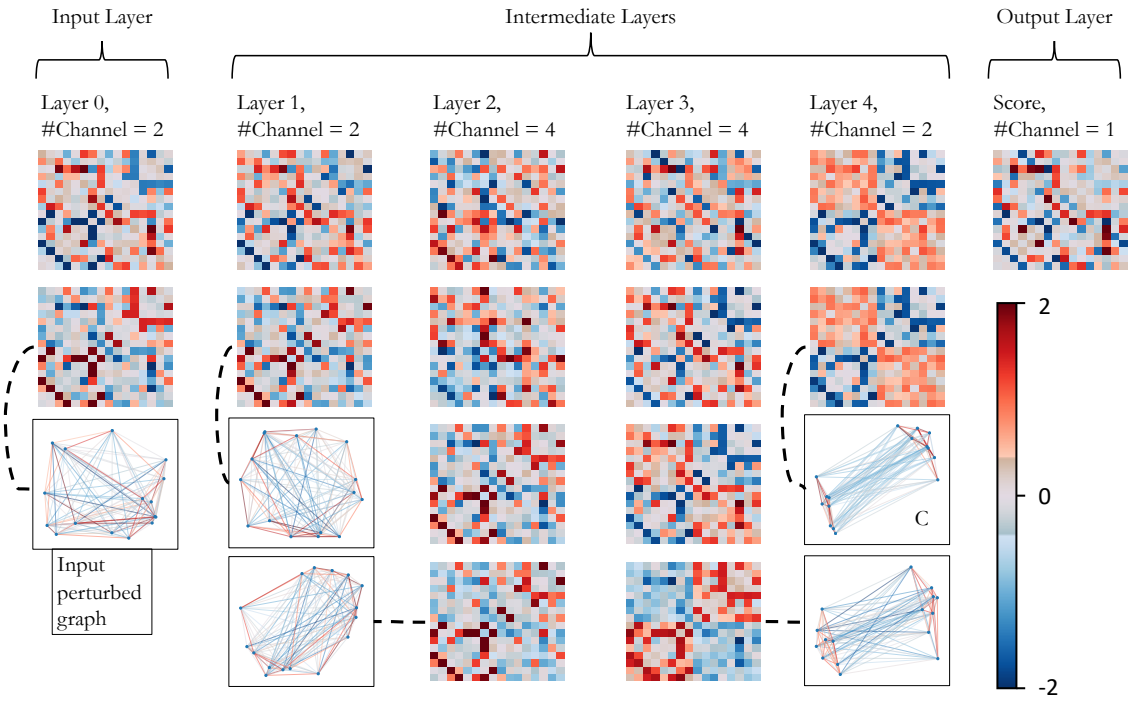


Figure 2: Visualization of channels for a pre-trained EDP-GNN model on the Community-small dataset. The model is trained with a single noise level $\sigma = 0.6$. The input is a community graph, but perturbed with Gaussian noise with $\sigma = 0.6$. The edge weights of each adjacency matrix are standardized 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.

networks (CNN) feature maps. Since channels of feature maps can be visualized as images in CNNs, we propose to visualize each channel of multi-channel adjacency matrices as a graph. The EDP-GNN layers should be able to map an input graph to intermediate graphs that possess interpretable semantics.

In Figure 2, we visualize the channels of intermediate adjacency matrices for a EDP-GNN model trained on the Community-small dataset. We observe that the model processes a perturbed community graph with no clearly visible structures to a graph with a structure of two equal-sized communities.

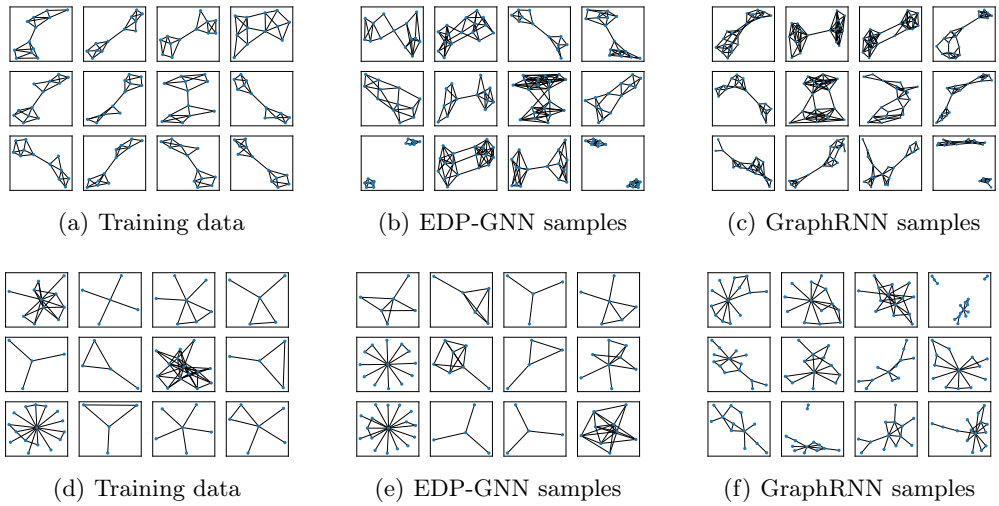
As implied by the training objective (3), the score network $\mathbf{s}_{\boldsymbol{\theta}}(\tilde{\mathbf{A}}, \sigma)$ can perfectly predict the ground truth score, *i.e.*, $\frac{\tilde{\mathbf{A}}-\mathbf{A}}{\sigma^2}$, if it can map the noise-perturbed graph $\tilde{\mathbf{A}}$ to the true (noise-free) graph $\mathbf{A}$ in some of the intermediate channels. Therefore, an ideal score network should be able to 1) understand the structure of a given graph, before 2) mapping a perturbed graph to the corresponding denoised graph. While previous GNNs are designed for the former task, EDP-GNN is especially capable of solving the latter one.

5.2.2 Ablation Studies

To verify the importance of intermediate adjacency matrices in EDP-GNN to be 1) learnable and 2) multi-channel, we conducted ablative studies on Community-small and Ego-small datasets. We switched on/off the two properties respectively, and provide the performance comparison in Tab. 3. Note that EDP-GNN is equivalent to vanilla GIN when intermediate adjacency matrices are single-channel and non-learnable. As shown in Tab. 3, both properties can improve the expressivity for score modeling, in the sense of reducing the training and test score matching losses. As expected, the performance is optimal when both properties are combined.

6 CONCLUSION

We propose a permutation invariant generative model for graphs based on the framework of score-based generative modeling. In particular, we implicitly define a per-



(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).

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).

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.

mutation invariant distribution over graph adjacency matrices by modeling the corresponding permutation equivariant score function and sampling with Langevin dynamics. For effective score modeling of graph distributions, we propose a new permutation equivariant GNN architecture, named EDP-GNN, leveraging trainable, multi-channel adjacency matrices as intermediate layers. Empirically, we demonstrate that EDP-GNNs are more expressive than vanilla GNNs on predicting edgewise features, as evidenced by better performance on the task of learning classic graph algorithms such as shortest paths. Moreover, we show our model can produce samples with quality comparable to existing state-of-the-art models. As one future direction, we hope to improve the scalability of our model by reducing the computational complexity, using techniques such as graph pooling (Ying et al., 2018).

Acknowledgements

This research was supported by Intel Corporation, Amazon AWS, TRI, NSF (#1651565, #1522054, #1733686), ONR (N00014-19-1-2145), AFOSR (FA9550-19-1-0024).

References

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

Batagelj, V. and Zaversnik, M. (2003). An o(m) algorithm for cores decomposition of networks. *arXiv preprint cs/0310049*.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2016). Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*.

Dobson, P. D. and Doig, A. J. (2003). Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, 330(4):771–783.

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. (2015). Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232.

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

Fout, A., Byrd, J., Shariat, B., and Ben-Hur, A. (2017). Protein interface prediction using graph convolutional networks. In *Advances in neural information processing systems*, pages 6530–6539.

Golomb, S. W. (1996). *Polyominoes: puzzles, patterns, problems, and packings*, volume 16. Princeton University Press.

Gómez-Bombarelli, R., Wei, J. N., Duvenaud, D., Hernández-Lobato, J. M., Sánchez-Lengeling, B., Sheberla, D., Aguilera-Iparraguirre, J., Hirzel, T. D., Adams, R. P., and Aspuru-Guzik, A. (2018). Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276.

Gong, L. and Cheng, Q. (2019). Exploiting edge features for graph neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9211–9219.

Gori, M., Monfardini, G., and Scarselli, F. (2005). A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE.

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

Grover, A., Zweig, A., and Ermon, S. (2018). Graphite: Iterative generative modeling of graphs. *arXiv preprint arXiv:1803.10459*.

Hamaguchi, T., Oiwa, H., Shimbo, M., and Matsumoto, Y. (2017). Knowledge transfer for out-of-knowledge-base entities: A graph neural network approach. *arXiv preprint arXiv:1706.05674*.

Hamilton, W., Ying, Z., and Leskovec, J. (2017). Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034.

Jin, W., Barzilay, R., and Jaakkola, T. (2018). Junction tree variational autoencoder for molecular graph generation. In *International Conference on Machine Learning*, pages 2328–2337.

Keriven, N. and Peyré, G. (2019). Universal invariant and equivariant graph neural networks. In *Advances in Neural Information Processing Systems*, pages 7090–7099.

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

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kipf, T., Fetaya, E., Wang, K.-C., Welling, M., and Zemel, R. (2018). Neural relational inference for interacting systems. In *International Conference on Machine Learning*, pages 2693–2702.

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

Leskovec, J., Chakrabarti, D., Kleinberg, J., Faloutsos, C., and Ghahramani, Z. (2010). Kronecker graphs: An approach to modeling networks. *Journal of Machine Learning Research*, 11(Feb):985–1042.

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

Li, Y., Zhang, L., and Liu, Z. (2018b). Multi-objective de novo drug design with conditional graph generative model. *Journal of cheminformatics*, 10(1):33.

Liao, R., Zhao, Z., Urtasun, R., and Zemel, R. S. (2019). Lanczosnet: Multi-scale deep graph convolutional networks. *arXiv preprint arXiv:1901.01484*.

Liu, J., Kumar, A., Ba, J., Kiros, J., and Swersky, K. (2019). Graph normalizing flows.

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

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2008). The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80.

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

Simonovsky, M. and Komodakis, N. (2018). Graphvae: Towards generation of small graphs using variational autoencoders. *arXiv preprint arXiv:1802.03480*.

Song, Y. and Ermon, S. (2019). Generative modeling by estimating gradients of the data distribution. *arXiv preprint arXiv:1907.05600*.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Watts, D. J. and Strogatz, S. H. (1998). Collective dynamics of 'small-world'networks. *nature*, 393(6684):440.

Xie, S., Kirillov, A., Girshick, R., and He, K. (2019). Exploring randomly wired neural networks for image recognition. *arXiv preprint arXiv:1904.01569*.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2018a). How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*.

Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.-i., and Jegelka, S. (2018b). Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pages 5449–5458.

Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., and Leskovec, J. (2018). Hierarchical graph representation learning with differentiable pooling. In *Advances in neural information processing systems*, pages 4800–4810.

You, J., Liu, B., Ying, Z., Pande, V., and Leskovec, J. (2018a). Graph convolutional policy network for goal-directed molecular graph generation. In *Advances in neural information processing systems*, pages 6410–6421.

You, J., Ying, R., Ren, X., Hamilton, W., and Leskovec, J. (2018b). Graphrnn: Generating realistic graphs with deep auto-regressive models. In *ICML*, pages 5694–5703.

A EXPERIMENTAL DETAILS

We implement our model using PyTorch (Paszke et al., 2019). The optimization algorithm is Adam (Kingma and Ba, 2014). Our code is available at `https://github.com/ermongroup/GraphScoreMatching`.

A.1 Hyperparameters

For the noise levels $\{\sigma_i\}_{i=1}^L$, we chose $L = 6$ and $\{\sigma_i\}_{i=1}^L = [1.6, 0.8, 0.6, 0.4, 0.2, 0.1]$. Empirically, we found those settings work well for all the generation experiments. Note that since all the edge weights in training data (*i.e.*, $\mathbf{A}_{i,j}$ in (2)) are either 0 or 1, $\sigma_L = 0.1$ is small enough for the quantizing operation (4) to prefectly recover the perturbed graph with high probability.

In the sampling process, we set the number of sampling steps for each noise level to be $T = 1000$. Apart from the coefficient ϵ in step size $\alpha_i = \epsilon \cdot \sigma_i^2/\sigma_L^2$ in Langevin dynamics, we added another scaling coefficient ϵ_s, since it is a common practice of applying Langevin dynamics. We chose the value of the hyper-parameters based on the MMD metrics on the validation set, which contains 32 samples from the training set.

$$\tilde{\mathbf{x}}_t \leftarrow \tilde{\mathbf{x}}_{t-1} + \frac{\alpha_i}{2}\mathbf{s}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}_{t-1}, \sigma_i) + \epsilon_s\sqrt{\alpha_i}\ \mathbf{z}_t$$

For the network architecture, we used 4 message-passing steps for each GIN, and stacked 5 EDP-GNN layers. The maximum number of channels of all EDP-GNN layer is 4. The maximum size of node features is 16.

A.2 Dataset

- **Community-small:** The graphs are constructed by two equal-sized communities, each of which is generated by E-R model (Erdos and Rényi, 1960), with $p = 0.7$. For each graph with N nodes, we randomly add $0.05N$ edges between the two communities. The range of total number of nodes per graph is $12 \leq N \leq 20$.
- **Ego-small:** One-hop ego graphs extracted from the Citeseer network (Sen et al., 2008). The range of node numbers per graph is $4 \leq N \leq 18$.

B PROPERTIES OF PERMUTATION INVARIANT FUNCTIONS

B.1 Permutation

Definition 1. *(Permutation Operation on Matrix) Let* $[N] \overset{def.}{=} \{1, \dots, N\}$. *Denote the set of permutations* $\pi : [N] \to [N]$ *as* Π_N. *The node permutation operation on a matrix* $\mathbf{A} \in \mathbb{R}^{N\times N}$ *is defined by* $\mathbf{A}_{i,j}^{[\pi]} = \mathbf{A}_{\pi(i),\pi(j)}$.

B.2 Permutation Invariant

Definition 2. *(Permutation Invariant Function) A function* f *with* $\mathbb{R}^{N\times N}$ *as its domain is permutation invariant i.f.f.* $\forall \mathbf{A} \in \mathbb{R}^{N\times N}, \forall \pi \in \Pi_N, \quad f(\mathbf{A}^{[\pi]}) = f(\mathbf{A})$.

B.3 Permutation Equivariant

Definition 3. *(Permutation Equivariant Function) A function* $\mathbf{s} : \mathbb{R}^{N\times N} \to \mathbb{R}^{N\times N}$ *is permutation equivariant i.i.f.* $\forall \mathbf{A} \in \mathbb{R}^{N\times N}, \forall \pi \in \Pi_N, \quad \mathbf{s}(A^{[\pi]}) = (\mathbf{s}(A))^{[\pi]}$.

B.4 Relationship between Permutation Invariance and Permutation Equivariance

Definition 4. *(Implicitly Defined Scalar Function) A function* $\mathbf{s} : \mathbb{R}^{N\times N} \to \mathbb{R}^{N\times N}$ *defines a gradient vector field on* $\mathbb{R}^{N\times N}$. *Veiw* $\mathbf{s}$ *as the gradient of a scalar value function* $f_{\mathbf{s}}(\mathbf{A}) : \mathbb{R}^{N\times N} \to \mathbb{R}$. *Define* $f_{\mathbf{s}}(\mathbf{A}) = \int_{\gamma[\mathbf{0},\mathbf{A}]} \langle \mathbf{s}(\mathbf{X}), \mathrm{d}\,\mathbf{X}\rangle_{\mathrm{F}} + C$, *where* $\mathbf{0} = \{0\}_{N\times N}$, $\gamma[\mathbf{0}, \mathbf{A}]$ *is any curve from* $\mathbf{0}$ *to* $\mathbf{A}$ *and* $C \in \mathbb{R}$ *is a constant.*

Under this definition, a vector function $\mathbf{s}$ defines a scalar function $f_{\mathbf{s}}$ implicitly.

Lemma 1. *(Permutation Invariance of Frobenius Inner Product) For any $A, B \in \mathbb{R}^{N\times N}$, the Frobenius inner product of A, B is $\langle \mathbf{A}, \mathbf{B}\rangle_{\mathrm{F}} = \sum_{i,j} A_{ij}B_{ij} = \mathrm{tr}(\mathbf{A}^T\mathbf{B})$. Frobenius inner product operation is permutation invariant,* i.e.*, $\forall \pi \in \Pi_N, \quad \langle \mathbf{A}^{[\pi]}, \mathbf{B}^{[\pi]}\rangle_{\mathrm{F}} = \langle \mathbf{A}, \mathbf{B}\rangle_{\mathrm{F}}$.*

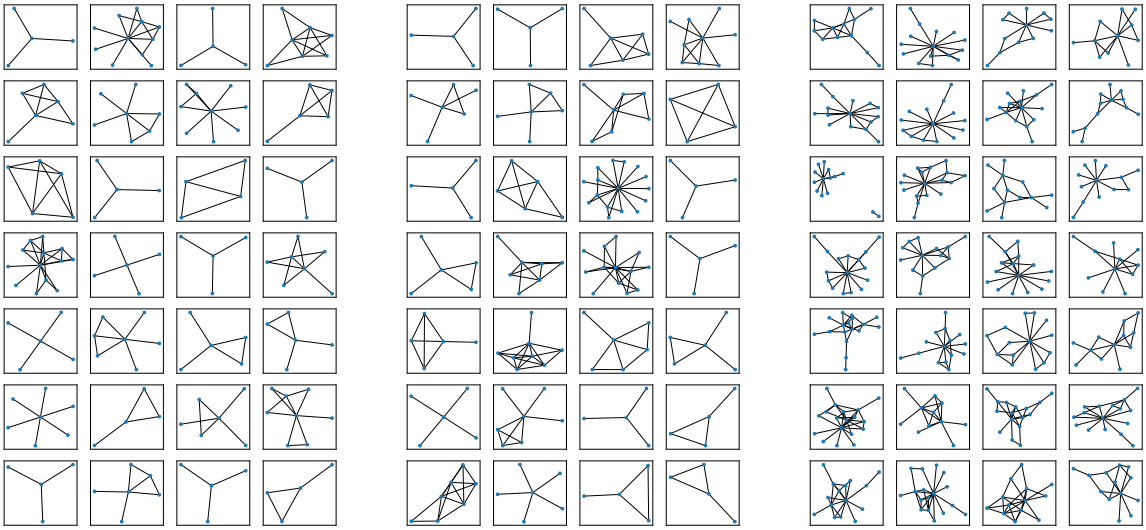
B.5 Proof of Theorem 1

Proof.

$$
\begin{aligned}
&\forall \mathbf{A} \in \mathbb{R}^{N\times N}, \forall \pi \in \Pi_N, \\
&f(\mathbf{A}^{[\pi]}) - f(\mathbf{0}^{[\pi]}) \\
=&\int_{\gamma[\mathbf{0}^{[\pi]}, \mathbf{A}^{[\pi]}]} \langle \mathbf{s}(\mathbf{X}), \mathrm{d}\,\mathbf{X}\rangle_{\mathrm{F}} \\
=&\int_{\gamma[\mathbf{0}, \mathbf{A}]} \langle \mathbf{s}(\mathbf{X}^{[\pi]}), \mathrm{d}\left(\mathbf{X}^{[\pi]}\right)\rangle_{\mathrm{F}} \\
=&\int_{\gamma[\mathbf{0}, \mathbf{A}]} \langle (\mathbf{s}(\mathbf{X}))^{[\pi]}, (\mathrm{d}\,\mathbf{X})^{[\pi]}\rangle_{\mathrm{F}} \\
=&\int_{\gamma[\mathbf{0}, \mathbf{A}]} \langle \mathbf{s}(\mathbf{X}), \mathrm{d}\,\mathbf{X}\rangle_{\mathrm{F}} \\
=&f(\mathbf{A}) - f(\mathbf{0}) \\
&\textit{i.e.} \quad f(\mathbf{A}^{[\pi]}) = f(\mathbf{A})
\end{aligned}
$$

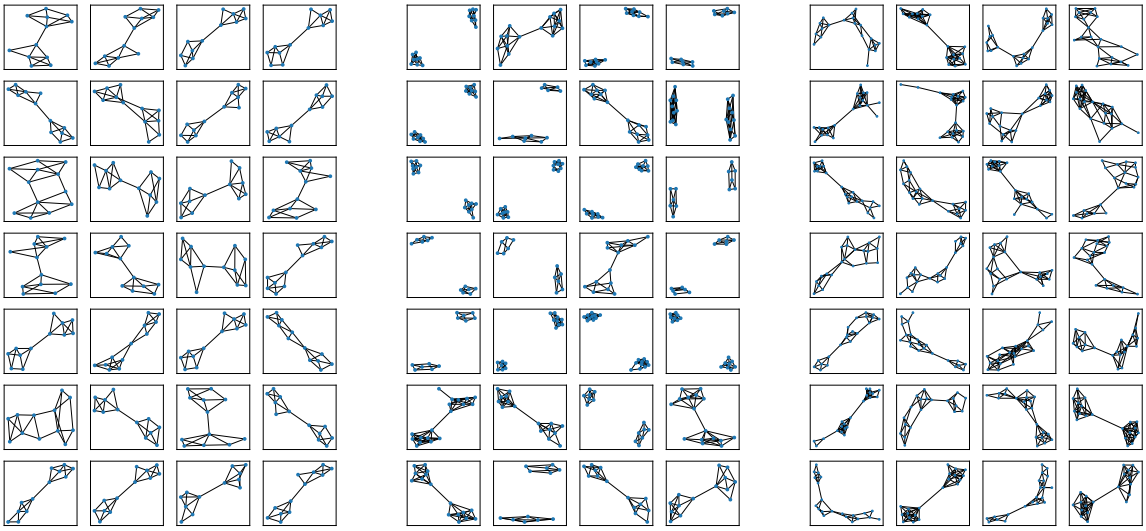
□

C EXTRA SAMPLES



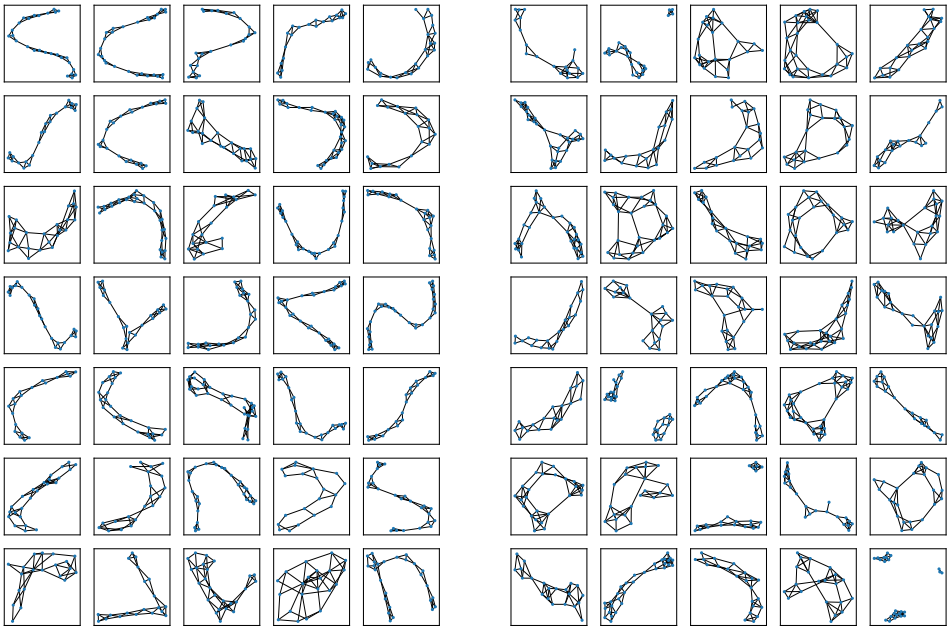
(a) Training data (b) EDP-GNN samples (c) GraphRNN samples

Figure 4: Extra samples from the training data, EDP-GNN, and GraphRNN, on Ego-small.



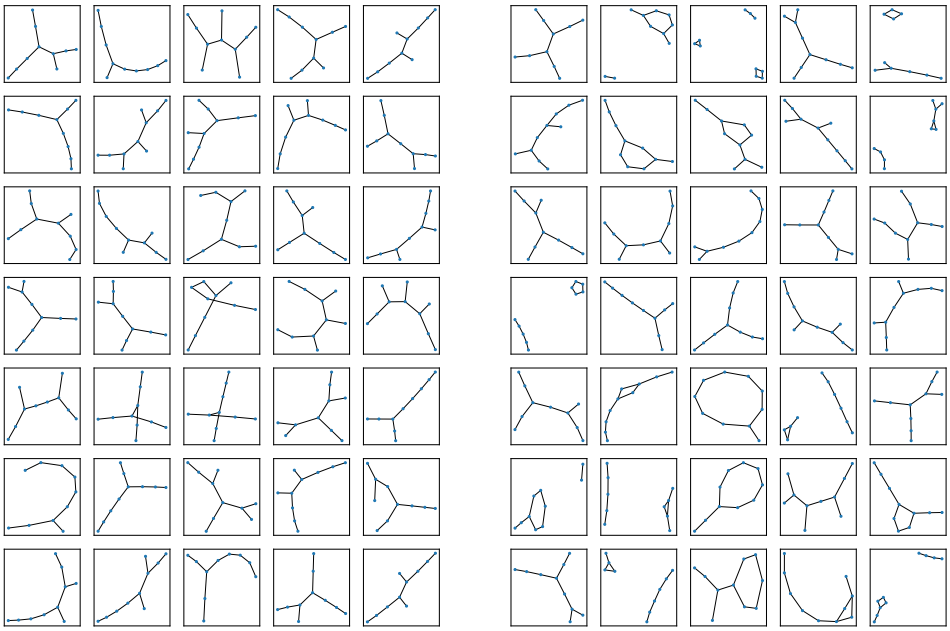
(a) Training data (b) EDP-GNN samples (c) GraphRNN samples

Figure 5: Extra samples from the training data, EDP-GNN, and GraphRNN, on Community-small.



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

Figure 6: Extra samples from the training data and EDP-GNN, on the Protein dataset (Dobson and Doig, 2003), with the number of node $20 \leq N \leq 30$.



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

Figure 7: Extra samples from the training data and EDP-GNN, on the Lobster graph dataset (Golomb, 1996), with the number of node $N = 10$.