Order Matters: Probabilistic Modeling of Node Sequence for Graph Generation

Xiaohui Chen [*1] **Xu Han** [*1] **Jiajing Hu** [1] **Francisco J. R. Ruiz** [2] **Liping Liu** [1]

Abstract

A graph generative model defines a distribution over graphs. One type of generative model is constructed by autoregressive neural networks, which sequentially add nodes and edges to generate a graph. However, the likelihood of a graph under the autoregressive model is intractable, as there are numerous sequences leading to the given graph; this makes maximum likelihood estimation challenging. Instead, in this work we derive the exact joint probability over the graph and the node ordering of the sequential process. From the joint, we approximately marginalize out the node orderings and compute a lower bound on the log-likelihood using variational inference. We train graph generative models by maximizing this bound, without using the ad-hoc node orderings of previous methods. Our experiments show that the log-likelihood bound is significantly tighter than the bound of previous schemes. Moreover, the models fitted with the proposed algorithm can generate high-quality graphs that match the structures of target graphs not seen during training. We have made our code publicly available at https://github.com/tufts-ml/graph-generation-vi.

1. Introduction

Random graphs have been a prominent topic in statistics and graph theory for decades. An early and influential model of random graphs is the Erdős–Rényi model (Erdős & Rényi, 1960). Since then, various models have been proposed to characterize different global statistics of graphs or networks in the real world (Watts & Strogatz, 1998; Nowicki & Snijders, 2001; Cai et al., 2016). However, these models are usually not designed for capturing local structures of a graph, such as bonds in a molecule graph.

Autoregressive generative models (You et al., 2018; Li et al., 2018; Liao et al., 2019; Dai et al., 2020; Goyal et al., 2020; Yuan et al., 2020; Shi et al., 2020) are designed to learn fine structures in graph data. These models generate a graph by sequentially adding nodes and edges. Since a graph is invariant to node permutations (Veitch & Roy, 2015), there are multiple sequences of actions leading to the same graph. When fitting an autoregressive model to data, a particular node ordering π of the graph G (called "generation order") is used to pin down a single generation sequence of G, such as depth-first search (DFS) or breadth-first search (BFS) ordering. The model is then fitted assuming the graph was generated under such ordering π. Autoregressive models of graphs typically use deep learning tools (Guo & Zhao, 2020), such as recurrent neural networks (RNNs), to learn flexible and complex patterns from data.

Choosing a specific ordering π does not rigorously correspond to maximum likelihood estimation (MLE). Indeed, to fit the parameters of an autoregressive model via MLE, we need the likelihood of G under the model. One approach for computing $p(G)$ is to sum over all possible node orderings π, $p(G) = \sum_\pi p(G, \pi)$. However, this approach presents some challenges. First, a generation sequence of G corresponds to multiple node orderings when G has non-trivial automorphisms (You et al., 2018; Liao et al., 2019), which require us to carefully derive the joint $p(G, \pi)$ from the model's distribution of generation sequences. Second, the marginalization is intractable in practice due to the number of terms in the sum. As a consequence, $p(G)$ cannot be easily obtained. This does not only make MLE intractable, but also implies that generative models cannot be evaluated in terms of log-likelihood. Instead, other evaluation metrics such as degree distribution are used, but these metrics exhibit some issues for complex graphs (Liu et al., 2019).

In this work, we provide a method to estimate the marginal log-likelihood, enabling standard statistical model checking and comparison. It also opens the door for other learning tasks that require the log-likelihood of graph data, such as density-based anomaly detection.

We aim at consolidating the foundation of autoregressive graph generative models. In particular, we examine two types of models: one that generates a graph through an

[*]Equal contribution [1]Department of Computer Science, Tufts University, Medford, MA, USA. [2]DeepMind, London, UK. Correspondence to: Xiaohui Chen <xiaohui.chen@tufts.edu>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

evolving graph sequence and one that generates an adjacency matrix. Then we derive the joint $p(G, \pi)$ from each type. Our analysis reveals a relationship between graph generation and graph automorphism.

To fit large graphs via MLE, we avoid the intractable marginalization by performing approximate posterior inference over the node ordering π. In particular, we use variational inference (VI) and maximize a lower bound of $\log p(G)$. We design a neural network that infers the probability over π for a given graph G. Thus, the generative model is trained with node orderings that are likely to generate G, avoiding the need to define ad-hoc orderings.

For evaluation, we estimate the graph log-likelihood via importance sampling. Our empirical study indicates that the variational lower bound is relatively tight. We also find that generative models fitted with the proposed method perform better than existing methods according to various metrics, including log-likelihood. Models trained with our method are able to generate new graphs with higher similarity to training graphs than existing approaches.

Contributions. Our main contributions are as follows:

- we give a rigorous definition of the probability of node orderings in autoregressive graph generative models;
- we analyze the relation between the calculation of graph probabilities and graph automorphism;
- we introduce VI to infer node orderings; and
- our training method with VI improves the performance of the model both quantitatively and qualitatively.

Related work. Autoregressive graph generation models have gained attention due to both the quality of generated graphs and their generation efficiency (You et al., 2018; Li et al., 2018; Liao et al., 2019; Dai et al., 2020; Shi et al., 2020). In these works, π is often decided by DFS or BFS, or it can be a specially designed canonical order. Liao et al. (2019) justify this approach by showing that these methods optimize a variational bound on $\log p(G)$. However, when the node orderings π are either randomly sampled from a uniform distribution or limited to a small range of canonical orders, these bounds are likely to be loose.

One model that considers a single canonical node ordering $\pi^\star$ is GraphGEN (Goyal et al., 2020). That is, for a given graph G, GraphGEN obtains $p(G)$ by considering that the graph was generated according to $\pi^\star$. However, when generating a graph from the model, GraphGEN does not guarantee the canonical order. This design raises a theoretical issue: the frequency of a generation sequence may not converge to the model's probability of that sequence.

2. Autoregressive Graph Generation

In Section 2.1, we introduce the two formulations of an autoregressive generative model—based on either a graph sequence or an adjacency matrix. In Section 2.2, we provide an explicit relationship between each formulation and the node ordering π to obtain the exact joint $p(G, \pi)$.

2.1. Problem definition

Let $V = \{1, \dots, n\}$ and E be the node set and edge set of a graph $G = (V, E)$ with $|V| = n$ nodes. A node ordering $\pi = (\pi_1, \dots, \pi_n)$ is a permutation of the elements in V. We consider G is unlabeled: permuting the nodes does not change the graph. The graph has a class $\mathcal{A}(G)$ of adjacency matrices corresponding to different node orderings—for each π, there is a unique adjacency matrix $\mathbf{A} \in \{0, 1\}^{n \times n}$ that indicates which nodes are connected. We only consider finite graphs without self-loops and multi-edges, so $\mathbf{A}$ is symmetric and its diagonal elements are zero. Let $\mathcal{G}$ denote the space of such graphs.

A generative model of unlabeled graphs defines a distribution $p(G)$ over $\mathcal{G}$. The distribution must be invariant to permutation of graph nodes. In this work, we focus on autoregressive generative models. We next review two formulations of autoregressive generative models.

The autoregressive model[1] by You et al. (2018); Liao et al. (2019); Shi et al. (2020); Goyal et al. (2020) operates with the adjacency matrix $\mathbf{A}$. In particular, the model generates a lower triangular matrix $\mathbf{L}$ by sequentially generating each row of $\mathbf{L}$. After every row is generated, it may stop with a special termination symbol, denoted by $\otimes$. Since an adjacency matrix $\mathbf{A} = \mathbf{L} + \mathbf{L}^\top$, each $\mathbf{L}$ uniquely determines $\mathbf{A}$ and vice-versa; thus $p(\mathbf{A}) = p(\mathbf{L})$ and

$$p(\mathbf{A}) = p(\otimes|\mathbf{L}) \prod_{t=2}^{n} p(L_{t,:}|\mathbf{L}_{1:(t-1)}). \tag{1}$$

Here, $\mathbf{L}_{1:(t-1)}$ denotes the submatrix formed from the first $t-1$ rows of $\mathbf{L}$, and $L_{t,:}$ is the t-th row of $\mathbf{L}$. The probability $p(L_{1,:}) = 1$ is left out here. The adjacency matrix $\mathbf{A}$ fully defines a graph G.

The deep generative model of graphs (DeepGMG) (Li et al., 2018) defines the sequential process as follows. It starts with a graph G_1 with one node, and at each step $t = 2, 3, \dots$, it obtains a graph G_t by adding a new node as well as some edges connecting the new node to the previously generated graph G_{t-1}. The probability of the sequence

[1] The formulation by Liao et al. (2019) generates graph nodes in batches, but it can also be expressed as an autoregressive model in this form. Similarly, GraphGEN (Goyal et al., 2020), which generates the sparse form of each row of $\mathbf{L}$, is also in this form.

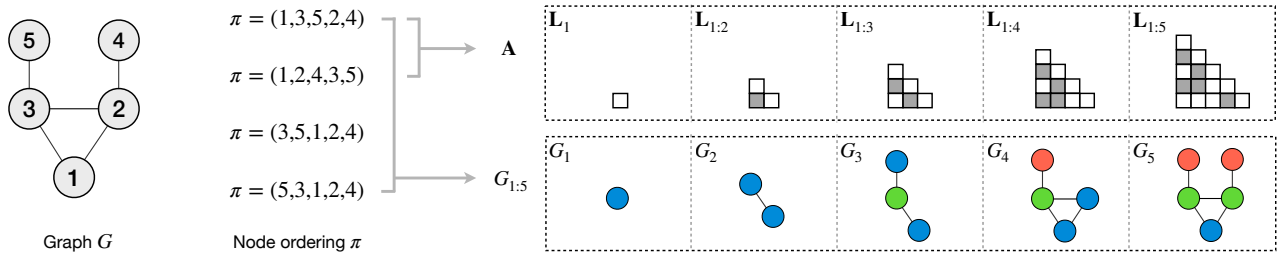

Figure 1. An overview of the relationship between the node ordering π and the adjacency matrix $\mathbf{A}$ or the graph sequence $G_{1:n}$. Given a graph G (left), several node orderings π (middle) specify the same adjacency matrix $\mathbf{A}$ or graph sequence $G_{1:n}$, so we cannot uniquely identify π from either $\mathbf{A}$ or $G_{1:n}$. The node orderings that give the same $\mathbf{A}$ give the same $G_{1:n}$, but not vice-versa. In the plot of $G_{1:n}$, for each subgraph G_t, nodes in the same orbit are labeled with the same color.

$G_{1:n} = (G_1, \ldots, G_n)$ is

$$p(G_{1:n}) = p(\otimes|G_n) \prod_{t=2}^{n} p(G_t|G_{t-1}). \tag{2}$$

The probability $p(G_1) = 1$ is left out here as well. Note that, after n steps, the graph G_n is the generated graph G.

A given graph G does not naturally have either a unique adjacency matrix $\mathbf{A}$ or a unique graph sequence $G_{1:n}$. Therefore, when fitting these models, we need to specify a node ordering π to pin down a single adjacency matrix $\mathbf{A}$ or sequence $G_{1:n}$. We depart from these two formulations and consider a formal treatment of the node ordering.

2.2. The generation order as a random variable

Here, we relate the sequential processes from Section 2.1 with the node ordering π. First, we consider the marginal likelihood $p(G)$. Under the first formulation, we obtain $p(G)$ by marginalizing over all adjacency matrices of G,

$$p(G) = \sum_{\mathbf{A} \in \mathcal{A}(G)} p(\mathbf{A}). \tag{3}$$

Under the second formulation, the marginalization is over all graph sequences that lead to G, i.e.,

$$p(G) = \sum_{G_{1:n}:\ G_n = G} p(G_{1:n}). \tag{4}$$

In both cases, the likelihood is intractable because the marginalization space is hard to specify—it involves finding all unique adjacency matrices or graph sequences (Liao et al., 2019). To obtain $p(G)$, many works use instead the node ordering π as the marginalization variable since the space of π is easier to characterize than that of $\mathbf{A}$ or $G_{1:n}$ for a graph G. To obtain $p(G, \pi)$, we need to clarify the relationship between $\mathbf{A}$ or $G_{1:n}$ and π, as we discuss next.

The sequential process from Section 2.2 generates an adjacency matrix or graph sequence; however in general we cannot identify π from either of these variables. To see this, consider first the relation between $\mathbf{A}$ and the node ordering π. Given the graph G, π determines $\mathbf{A}$ because the t-th row of $\mathbf{A}$ correponds to node π_t. However, the converse is not necessarily true: a matrix $\mathbf{A}$ corresponds to multiple node orderings if G has non-trivial automorphism (Liao et al., 2019). We provide an example in Figure 1, where each of the first two node orderings ($\pi = (1, 3, 5, 2, 4)$ and $\pi = (1, 2, 4, 3, 5)$) determines $\mathbf{A}$, but we cannot uniquely identify one of them from $\mathbf{A}$ (in particular, we cannot distinguish the node pairs $(2, 4)$ and $(3, 5)$). The same is true for the graph sequence $G_{1:n}$: a node ordering π defines a graph sequence $G_{1:n}$, but not vice-versa (see Figure 1).

Similarly, the relation between $\mathbf{A}$ and $G_{1:n}$ is not unique. An adjacency matrix $\mathbf{A}$ determines a graph sequence $G_{1:n}$, but a graph sequence does not determine a unique $\mathbf{A}$. As an example, in Figure 1 all four node orderings generate the same $G_{1:n}$, but the last two node orderings determine two adjacency matrices different from the shown matrix $\mathbf{A}$.

In summary, (G, π) determines $\mathbf{A}$, which determines $G_{1:n}$, but the reverse is not true in general. This implies that *an autoregressive generative model (which generates $\mathbf{A}$ or $G_{1:n}$) does not specify a distribution over π.*

We next make π a random variable and formally specify the joint $p(G, \pi)$. Given the graph $G = (V, E)$, let $\Pi[\mathbf{A}]$ be the set of all possible node orderings π that give the same adjacency $\mathbf{A}$; similarly, let $\Pi[G_{1:n}]$ be the set of all node orderings that give the same graph sequence $G_{1:n}$, i.e.,

$$\Pi[\mathbf{A}] = \{\pi : A_{\pi_i, \pi_j} = \mathbb{1}[(\pi_i, \pi_j) \in E], \forall i, j \in V\}$$
$$\Pi[G_{1:n}] = \{\pi : G[\pi_{1:t}] = G_t, \forall t = 1, \ldots, n\}.$$

Here, $\mathbb{1}[\cdot]$ is 1 or 0 depending on whether the condition in the bracket is true or false, and $G[\pi_{1:t}]$ is the induced subgraph of G from the first t nodes in the ordering π. Then we let the conditional distribution $p(\pi|\mathbf{A})$ be uniform, i.e.,

$$p(\pi|\mathbf{A}) = \frac{1}{|\ \Pi[\mathbf{A}]\ |}. \tag{5}$$

The set $\Pi[\mathbf{A}]$ turns out to be the set of automorphisms[2] of the graph G. This is because every node ordering $\pi \in \Pi[\mathbf{A}]$ permutes rows and columns of $\mathbf{A}$ but does not change $\mathbf{A}$; that is, each π creates an automorphism. Therefore, obtaining $p(\pi|\mathbf{A})$ amounts to finding the number of automorphisms of a graph. Fortunately, this is a well-studied classic problem in graph theory. The time complexity of computing $|\Pi[\mathbf{A}]|$ is $\exp\left(\mathcal{O}(\sqrt{n \log n})\right)$ (Beals et al., 1999). The Nauty package (McKay & Piperno, 2013) uses various heuristics and can efficiently find this number for most graphs. In practice, it can compute $|\Pi[\mathbf{A}]|$ for a graph with thousands of nodes within 10^{-3} seconds.

For the formulation with graph sequences, the analysis is more involved. We define the conditional $p(\pi|G_{1:n})$ as a uniform distribution,

$$p(\pi|G_{1:n}) = \frac{1}{\left|\, \Pi[G_{1:n}] \,\right|}. \tag{6}$$

We discuss below how to obtain $|\Pi[G_{1:n}]|$ in practice, but first we formally specify the joint $p(G, \pi)$ and the likelihood $p(G)$. The joint can be obtained from $p(\mathbf{A})$ or $p(G_{1:n})$ as

$$p(G, \pi) = \frac{1}{\left|\, \Pi[\mathbf{A}] \,\right|} p(\mathbf{A}) = \frac{1}{\left|\, \Pi[G_{1:n}] \,\right|} p(G_{1:n}). \tag{7}$$

(This expression assumes that $\mathbf{A} \in \mathcal{A}(G)$ and that $G_n = G$.) The marginal likelihood $p(G)$ of a graph can be obtained by marginalizing out the node ordering π from Eq. 7,

$$p(G) = \sum_{\pi} p(G, \pi). \tag{8}$$

Obtaining $p(G)$ from Eq. 8 is easier than from Eq. 3 or Eq. 4 because the marginalization space is easier to characterize, but it remains intractable because of the large number of terms in the sum. In Section 3, we derive a variational bound on $p(G)$ by approximating the posterior distribution $p(\pi|G)$, for which we use the definition of the joint in Eq. 7.

Obtaining $|\Pi[G_{1:n}]|$. We now discuss the practical calculation of $|\Pi[G_{1:n}]|$. Like $|\Pi[\mathbf{A}]|$, it is also closely related to graph automorphism. Let $\mathrm{Aut}(G)$ denote the set of all automorphisms of G, then the *orbit* of a node $u \in V$ is $r(G, u) = \{v \in V : \exists f \in \mathrm{Aut}(G), v = f(u)\}$ (Godsil & Royle, 2001). Intuitively, the orbit of u contains all nodes that are "symmetric" to u. In Figure 1, the orbit of node 3 is $\{2, 3\}$, and the orbit of node 5 is $\{4, 5\}$. The theorem below expresses $|\Pi[G_{1:n}]|$ in terms of the cardinality of the orbits produced during the sequential generative process.

Theorem 1. *For a graph sequence $G_{1:n}$, we have*

$$\left|\, \Pi[G_{1:n}] \,\right| = \prod_{t=1}^{n} \left|\, r(G_t, \pi_t) \,\right|. \tag{9}$$

[2] A function $f : V \to V$ is an automorphism of $G = (V, E)$ if $(u, v) \in E \iff (f(u), f(v)) \in E$.

We show an example before providing the proof. Suppose that G_n is the complete graph with n nodes, then each G_t in the sequence is a complete graph with t nodes. Applying the theorem with $r(G_t, \pi_t) = t$ gives $|\Pi[G_{1:n}]| = n!$, which means that all $n!$ permutations use the same graph sequence.

Proof. The proof of the theorem needs the following lemma, whose proof is in Appendix A.2.

Lemma 1. *Let $G[V\backslash\{u\}]$ and $G[V\backslash\{v\}]$ respectively denote the subgraphs induced by $V\backslash\{u\}$ and $V\backslash\{v\}$, then u and v are in the same orbit if and only if $G[V\backslash\{u\}]$ and $G[V\backslash\{v\}]$ are isomorphic.*

We prove Eq. 9 by induction. Let $\pi \in \Pi[G_{1:n}]$, and consider the number of node orderings that give the same graph sequence as π. When $n = 1$, there is only one node in the graph, and then the base case is true: $|\Pi[G_1]| = |r(G_1, 1)| = 1$. Then, we show the induction rule $|\Pi[G_{1:n}]| = |\Pi[G_{1:(n-1)}]| \cdot |r(G_n, \pi_n)|$. If a node ordering π' of G_n gives the same graph sequence $G_{1:n}$ as π, then nodes π'_n and π_n must be in the same orbit by the lemma. There are $|r(G_n, \pi_n)|$ choices of π'_n. Then, consider the number of choices for $\pi'_{1:(n-1)}$. Since removing π_n and removing π'_n give two isomorphic graphs, $\pi'_{1:(n-1)}$ can take any node ordering in $\Pi[G_{1:(n-1)}]$ and thus has $|\Pi[G_{1:(n-1)}]|$ possible values. Together, π' has $|\Pi[G_{1:n}]|$ possible values, which implies the induction rule. □

To compute $r(G_t, \pi_t)$, we need to identify the orbit of the node π_t, which can be expensive for some graphs. Thus, we resort instead to an approximation of $r(G_t, \pi_t)$ that ultimately results in a lower bound of $p(G)$. The approximation is based on the color refinement algorithm (1-Weisfeiler-Lehman), which approximately obtains the orbit of a node. The algorithm uses node colors to partition nodes and always assigns the same color to nodes in the same orbit (Arvind et al., 2017). Let $\mathbf{c} = \mathrm{CR}(G)$ be node colors from the color refinement algorithm; then $r_{\mathrm{CR}}(G, u) = \{v \in V : c_v = c_u\} \supseteq r(G, u)$ (the two sets are equal for most cases since the color refinement algorithm is very effective in practice). Then, we can use the result of the algorithm to obtain a bound of Eq. 9,

$$\beta(G_{1:n}) \triangleq \prod_{t=1}^{n} |r_{\mathrm{CR}}(G_t, \pi_t)| \geq \left|\, \Pi[G_{1:n}] \,\right|. \tag{10}$$

This implies a bound on the joint $p(G, \pi)$ from Eq. 7,

$$\widehat{p}(G, \pi) \triangleq \frac{1}{\beta(G_{1:n})} p(G_{1:n}) \leq p(G, \pi). \tag{11}$$

This bound is tight in practice because of the effectiveness of the color refinement algorithm. In Section 3, we optimize

a variational bound on the marginal $\sum_\pi \widehat{p}(G,\pi) \approx p(G)$, but we write $p(G)$ and $p(G,\pi)$ for simplicity.

Can we avoid the marginalization by using a single generation order for a graph? GraphGEN (Goyal et al., 2020) defines a single canonical node ordering $\pi^\star$ for a given graph G. Then, there is only one adjacency matrix $\mathbf{A}^\star$ corresponding to $\pi^\star$, and GraphGEN defines $p(G) = p(\mathbf{A}^\star)$, therefore avoiding the marginalization over π. However, GraphGEN does not restrict the generation order when sampling from the model; in fact there is not a straightforward way to control the generation order because the canonical order is computed retrospectively after G is generated. As a result, a sample from GraphGEN may be generated with a node ordering that is different from the canonical order of the resulting graph. Thus, the sampling probability of G is likely to be inconsistent with the probability $p(G)$ that the model assigns to G. That is, the sampling frequency of G will not converge to the model's $p(G)$, which is a severe problem for a statistical model. To estimate how different the sampling and the model probabilities are, we tested the generation procedure of GraphGEN, and we found that only 9.1% of the generated graphs use the canonical order that is used for the calculation of $p(G)$ during training.

3. Training a Generative Model using VI

Here we present a method to fit an autoregressive graph generation model that does not rely on any constraints on the node ordering. We use the notation $p_\theta(G,\pi)$ to explicitly indicate that the joint depends on the parameters θ of the generative model—either $p_\theta(\mathbf{A})$ or $p_\theta(G_{1:n})$. For moderately large graphs, the MLE of θ is computationally intractable because the marginalization of π from Eq. 8 involves $n!$ terms; we sidestep this issue with a VI method (Blei et al., 2017) that maximizes a lower bound on $\log p_\theta(G)$.

The variational lower bound $L(\theta,\phi,G) \le \log p_\theta(G)$ is

$$L(\theta,\phi,G) = \mathbb{E}_{q_\phi(\pi|G)}\left[\log p_\theta(G,\pi) - \log q_\phi(\pi|G)\right]. \quad (12)$$

Here $q_\phi(\pi|G)$ is a variational distribution to approximate the posterior $p_\theta(\pi|G)$. Its parameters are denoted by ϕ. We fit the model parameters θ and the variational parameters ϕ by maximizing Eq. 12 w.r.t. both parameters. We discuss the form of the variational distribution $q_\phi(\pi|G)$ in Section 3.1 and the optimization algorithm in Section 3.2.

3.1. The variational distribution

The variational distribution $q_\phi(\pi|G)$ approximates the intractable posterior $p_\theta(\pi|G)$. To obtain a good approximation, we let $q_\phi(\pi|G)$ incorporate both graph topological information as well as the information from partially generated graphs according to the order π. We use a Recurrent Ordering Structure (ROS) to specify $q_\phi(\pi|G)$,

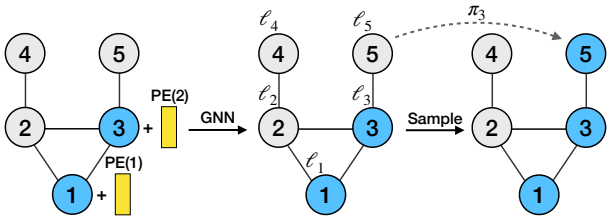

Figure 2. Illustration of the sampling procedure from the variational distribution. (Left) To sample node $\pi_3 = 5$ given $\pi_1 = 1$ and $\pi_2 = 3$, we first augment the initial node features with positional embeddings using Eq. 15. (Middle) The GNN obtains the logits for each node using Eq. 16. (Right) We sample π_3 from the categorical distribution (Eq. 14).

$$q_\phi(\pi|G) = \prod_{t=1}^{n} q_\phi(\pi_t|G,\pi_{1:(t-1)}). \quad (13)$$

At each step, the distribution of the t-th node π_t depends on both G and the partial order $\pi_{1:(t-1)}$. In particular, the conditional $q_\phi(\pi_t|G,\pi_{1:(t-1)})$ is a categorical distribution over π_t; we denote its logits by $\{\ell_k^t\}$, then

$$q_\phi(\pi_t|G,\pi_{1:(t-1)}) = \frac{\exp\{\ell_{\pi_t}^t\}}{\sum_{k\notin\pi_{1:(t-1)}}\exp\{\ell_k^t\}},\ \pi_t \notin \pi_{1:(t-1)}. \quad (14)$$

The logits are functions of $(G,\pi_{1:(t-1)})$. We use a graph neural network (GNN) as the recurrent unit that outputs the logits $\{\ell_k^t\}$ of the conditional $q_\phi(\pi_t|G,\pi_{1:(t-1)})$, since GNNs are powerful tools to extract information from graphs. The input of a GNN usually consists of the graph G and its node features; in our case the input is $\pi_{1:(t-1)}$ and G. To encode $\pi_{1:(t-1)}$ into an initial set of node features $\{\mathbf{h}_1^t,...,\mathbf{h}_n^t\}$, we use a positional embedding $\mathrm{PE}(\cdot)$ (Vaswani et al., 2017), such that

$$\mathbf{h}_j^t = \begin{cases} \mathbf{h}_0 + \mathrm{PE}(t), & \text{if } j = \pi_{t'} \text{ for } t' < t, \\ \mathbf{h}_0, & \text{otherwise.} \end{cases} \quad (15)$$

Here, $\mathbf{h}_0$ is a learnable vector used globally for all steps and nodes. (If the graph data contains node features, we can use these node features to replace $\mathbf{h}_0$.) Then, the GNN computes the logits for all nodes.

$$(\ell_1^t,\ldots,\ell_n^t) = \mathrm{GNN}_\phi(G,(\mathbf{h}_1^t,\ldots,\mathbf{h}_n^t)). \quad (16)$$

Only logits for nodes not in $\pi_{1:(t-1)}$ are used for the calculation of (14). Figure 2 illustrates the process to sample from the conditional $q_\phi(\pi_t|G,\pi_{1:(t-1)})$.

The choice of the specific GNN is flexible. In our experiments, the graph attention network (GAT) (Veličković et al.,

Algorithm 1 VI algorithm for training a graph model based on the adjacency matrix $\mathbf{A}$

Input: Dataset of graphs $\mathcal{G} = \{G_1, \ldots, G_n\}$, model p_θ, variational distribution q_ϕ, sample size S
Output: Learned parameters θ and ϕ
repeat
  for $G \in \mathcal{G}$ **do**
    Sample $\pi^{(1)}, \ldots, \pi^{(S)} \overset{\text{iid}}{\sim} q_\phi(\pi|G)$
    Obtain $\mathbf{A}^{(s)}$ from $(G, \pi^{(s)})$
    Set $p_\theta(G, \pi^{(s)}) = \frac{1}{|\Pi[\mathbf{A}^{(s)}]|} p_\theta(\mathbf{A}^{(s)})$
    Compute $\nabla_\phi \leftarrow \nabla_\phi L(\theta, \phi, G)$
    Compute $\nabla_\theta \leftarrow \nabla_\theta L(\theta, \phi, G)$
    Update ϕ, θ using the gradients ∇_ϕ, ∇_θ
  end for
until convergence of the parameters (θ, ϕ)

2017) performed better than the graph convolutional network (GCN) (Wu et al., 2019) and the approximate personalized propagation of neural predictions (APPNP) (Klicpera et al., 2018). All results in Section 4 use the GAT.

3.2. Maximizing the variational lower bound

To maximize the lower bound $L(\theta, \phi, G)$ in Eq. 12, we need its gradients w.r.t. both θ and ϕ, which are intractable. We obtain the gradient w.r.t. θ via Monte Carlo estimation. We obtain the gradient w.r.t. ϕ using the score function estimator (Williams, 1992; Carbonetto et al., 2009; Paisley et al., 2012; Ranganath et al., 2014). The estimators are obtained with S samples $\pi^{(s)} \sim q_\phi(\pi|G)$ for $s = 1, \ldots, S$, yielding

$$\nabla_\theta L(\theta, \phi, G) \approx \frac{1}{S} \sum_{s=1}^{S} \nabla_\theta \log p_\theta(G, \pi^{(s)}), \tag{17}$$

$$\nabla_\phi L(\theta, \phi, G) \approx \frac{1}{S} \sum_{s=1}^{S} \Big[\log p_\theta(G, \pi^{(s)}) - \log q_\phi(\pi^{(s)}|G) \Big] \nabla_\phi \log q_\phi(\pi^{(s)}|G). \tag{18}$$

Eq. 17 shows that the parameters θ of the model are optimized under node sequences π sampled from the approximate posterior. That is, fitting the model does not require to define ad-hoc orderings π; rather, the (approximately) most likely node orderings are used. As a comparison, a model trained with uniformly distributed random node orderings can be seen as using a uniform variational distribution, which in turn corresponds to a looser log-likelihood bound.

Although the score function estimator may exhibit large variance in general, in our experiments we found that this does not represent an issue. In fact, $S = 4$ samples were enough and allowed for stable optimization of the objective (see Appendix A.1). We leave other gradient estimation techniques (Mohamed et al., 2019) for future work.

We present the training procedure in Algorithm 1. The algorithm can be applied to many autoregressive models operating with the adjacency matrix $\mathbf{A}$, such as GraphRNN and GraphGEN. For models that operate with the graph sequence instead, such as DeepGMG, we only need to extract the graph sequence $G^{(s)}_{1:n}$ from each $(G, \pi^{(s)})$ and set $p_\theta(G, \pi^{(s)}) = \frac{1}{|\Pi[G^{(s)}_{1:n}]|} p_\theta(G^{(s)}_{1:n})$.

Running time. To form the gradient estimators, each of the S Monte Carlo samples requires n evaluations of the GNN output, each taking $\mathcal{O}(|E|)$. For most graphs, the complexity of the gradient computation is dominated by these terms and is therefore $\mathcal{O}(Sn|E|)$. Counting automorphisms only takes a small fraction of the running time in practice. Similarly, the approximation of $|\Pi[G_{1:n}]|$ also takes a small fraction of the running time. The resulting $\mathcal{O}(Sn|E|)$ complexity is a limitation of the proposed algorithm, and hence it is hard to scale to large graphs. However, since it provides better results than existing approaches (see Section 4), our algorithm can still be preferable for applications that are not sensitive to the training time. We leave for future work the exploration of ways to improve the computational efficiency, such as proposing the node ordering π in one shot.

4. Experiments

In this section, we design a set of experiments to investigate: (i) the tightness of the variational lower bound, (ii) the performance of a model fitted with the proposed method based on VI, (iii) the quality of the approximate posterior learned by the variational distribution, and (iv) the quality of graphs generated with the fitted model.

4.1. Experimental setup

Datasets. We use 6 datasets: (1) *Community-small*: 500 community graphs with $12 \leq |V| \leq 20$. Each graph has two communities generated by the model of Erdős & Rényi (1960). (2) *Citeseer-small*: 200 subgraphs with $5 \leq |V| \leq 20$, extracted from Citeseer network (Sen et al., 2008) using random walk. (3) *Enzymes*: 563 protein graphs from BRENDA database (Schomburg et al., 2004) with $10 \leq |V| \leq 125$. (4) *Lung*: 400 chemical graphs with $6 \leq |V| \leq 50$, sampled from Kim et al. (2018). (5) *Yeast*: 400 chemical graphs with $5 \leq |V| \leq 50$, sampled from Kim et al. (2018). (6) *Cora*: 400 subgraphs with $9 \leq |V| \leq 97$, extracted from the Cora network (Sen et al., 2008) using random walk.

Methods. We choose three recent graph generative models, DeepGMG (Li et al., 2018), GraphRNN (You et al., 2018), and GraphGEN (Goyal et al., 2020). We use their original training methods with default hyperparameters as baselines, and compare them with the proposed VI method. For our method, we use the Nauty package (McKay & Piperno, 2013) to compute $|\Pi[A]|$ and the color refinement algo-

Table 1. Approximate test log-likelihood and variational lower bound (ELBO) of different graph generation models. For each model, we compare the default training algorithm with our method based on VI; the table shows that VI improves the model's predictive performance. Moreover, the variational bound is relatively tight. We used paired t-test to compare the results; the numbers in bold indicate that the method is better at the 5% significance level.

		Community-small log-like/ELBO	Citeseer-small log-like/ELBO	Enzymes log-like/ELBO	Lung log-like/ELBO	Yeast log-like/ELBO	Cora log-like/ELBO
DeepGMG	uniform	-206.2/-303.9	**-60.9/-67**	-281.9/-290.8	**-146.7/-225.7**	-115.1/128.9	-283.7/-295.2
	VI [ours]	**-124.8/-131.8**	**-59.6/-65.6**	**-145.8/-156.2**	**-146.1/-224.6**	**-105.4/-115.7**	**-227/-247.2**
GraphRNN	uniform	-154.6/-157.6	-101.9/-105.7	-340.3/-349.1	-232.4/ -242.2	-189.3/-200.1	-380.6/-401.8
	VI [ours]	**-53.7/-59.9**	**-89.6/-93.2**	**-274.9/-282.8**	**-155.9/-175.8**	**-109.1/-133.7**	**-345.3/-358.3**
GraphGEN	DFS	-263.74/NA	-73.0/NA	-574.2/NA	-140.1/NA	**-66.46/NA**	-199.5/NA
	VI [ours]	**-26.6/-35.0**	**-64.3/-71.1**	**-189.7/-213.8**	**-117.3/-125.5**	**-64.98/-72.39**	**-143.6/-152.3**

Table 2. Graph quality on the considered datasets (MMD on three metrics). Models fitted with VI tend to produce higher-quality samples.

		Community-small			Citeseer-small			Enzymes		
		Deg.	Clus.	Orbit	Deg.	Clus.	Orbit	Deg.	Clus.	Orbit
DeepGMG	uniform	0.2	0.978	0.40	0.052	0.06	**0.005**	1.51	0.95	0.29
	VI [ours]	**0.178**	**0.921**	**0.338**	**0.028**	**0.014**	**0.005**	**1.01**	**0.48**	**0.27**
GraphRNN	BFS	0.034	0.11	0.009	0.016	**0.05**	0.004	0.03	0.085	0.043
	uniform	0.096	0.091	0.021	**0.009**	0.09	0.003	0.042	0.104	0.074
	VI [ours]	**0.018**	**0.01**	**0.008**	0.08	**0.05**	**0.002**	**0.015**	**0.067**	**0.02**
GraphGEN	DFS	0.695	0.931	0.178	0.047	**0.032**	0.017	0.716	0.456	0.078
	VI [ours]	**0.143**	**0.248**	**0.068**	**0.032**	0.078	**0.008**	**0.346**	**0.440**	**0.020**
		Lung			Yeast			Cora		
		Deg.	Clus.	Orbit	Deg.	Clus.	Orbit	Deg.	Clus.	Orbit
DeepGMG	uniform	0.206	**0.023**	0.224	0.547	0.242	0.470	**0.35**	0.27	0.11
	VI [ours]	**0.189**	**0.023**	**0.2**	**0.324**	**0.118**	**0.258**	0.36	**0.22**	**0.04**
GraphRNN	BFS	0.103	0.301	0.043	0.512	0.153	0.026	1.125	1.002	0.427
	uniform	1.213	**0.002**	0.081	0.746	0.351	0.070	0.188	0.206	0.200
	VI [ours]	**0.074**	0.060	**0.004**	**0.097**	**0.092**	**0.005**	**0.066**	**0.171**	**0.052**
GraphGEN	DFS	0.049	0.017	**0.000**	0.014	**0.003**	**0.000**	0.099	0.167	0.122
	VI [ours]	**0.022**	**0.008**	**0.000**	**0.012**	**0.003**	**0.000**	**0.056**	**0.103**	**0.069**

rithm to approximate $|\Pi[G_{1:n}]|$, and we parameterize the variational distribution with a GAT (Veličković et al., 2017) with 3 layers, 6 attention heads, and residual connections.

4.2. Predictive performance in terms of log-likelihood

Here we compare the different methods in terms of the log-likelihood on test data. We approximate the log-likelihood using importance sampling (Murphy, 2012). We use the variational distribution $q_\phi(\pi|G)$ as the proposal distribution and draw L samples $\{\pi^{(l)}\}$ from it. The importance sampling approximation of $\log p(G)$ is

$$\log p_\theta(G) \simeq \log \Big(\frac{1}{L} \sum_{l=1}^{L} \frac{p_\theta(G, \pi_l^{(l)})}{q_\phi(\pi^{(l)}|G)} \Big). \qquad (19)$$

Here $\pi^{(l)} \sim q_\phi(\pi|G)$ for $l = 1, \ldots, L$. The estimation is unbiased only when L approaches infinity; nevertheless, we found that $L = 1{,}000$ gives an accurate estimation (see Appendix A.3).

For our method, we use the learned $q_\phi(\pi|G)$ distribution as the proposal in the importance sampling approximation. For DeepGMG and GraphRNN, we use a uniform proposal $q_\phi(\pi|G)$, because these methods are trained with node orderings sampled from the uniform distribution (as mentioned before, this is equivalent to using a uniform variational distribution). We use $L = 1{,}000$ samples for each graph in the test set, except for GraphGEN, for which we only use the canonical order $\pi^\star$ to estimate the log-likelihood.

The results are in Table 1. We compare the results from each baseline and from our approach using a paired t-test at the 5% significance level. We see that the proposed VI method exhibits better predictive performance on most datasets, and the improvements are often very significant. To assess the quality of the variational lower bound, we also show its value in Table 1 (the bound was estimated with 1000 samples from $q_\phi(\pi|G)$). We can see that the bound is relatively tight for most cases. These results indicate that our training procedure based on VI can significantly improve the performance of a graph generative model.

On the Yeast dataset, the result of the VI approach is very close to the DeepGMG baseline. We checked the node orderings sampled from the learned variational distribution and observed that they are very similar to DFS orders. We hypothesize that, for this dataset, the posterior $p_\theta(\pi|G)$ is higher for DFS orders, and that $q_\phi(\pi|G)$ can find this. On the Community-small dataset, the gap with the baseline is

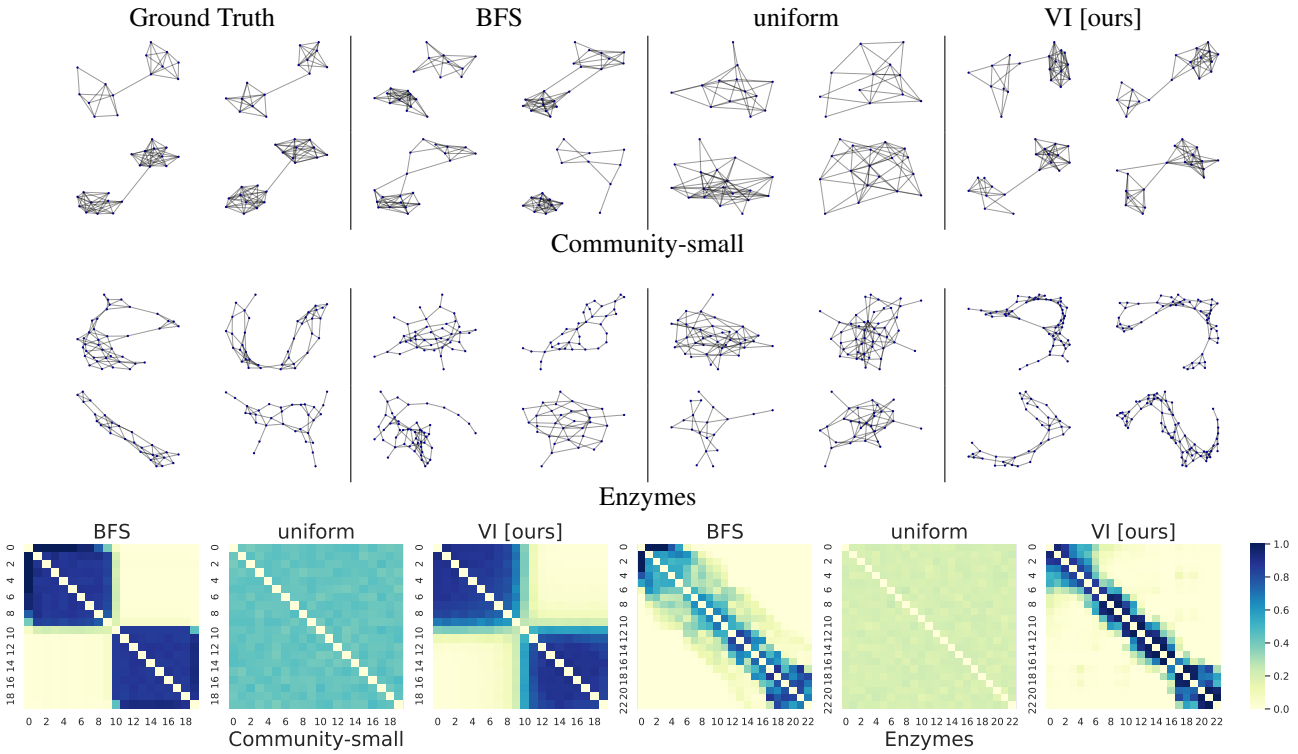

Figure 3. (Top) Graph samples from different models trained on Community-small and Enzymes. The model fitted with VI learns to generate graphs with the same structural patterns as the real data. (Bottom) Averaged adjacency matrices for a graph with different samples from the node ordering. Our VI approach uncovers the underlying structure of the graph.

much larger; this is because the graphs in this dataset have a special structure that always connects two communities with one edge. The variational distribution seems to be able to exploit this structure to improve the model fitting. For the Citeseer-small and Cora datasets, the gap is smaller—these datasets are generated from random walks, so the graphs have less structure for the VI algorithm to exploit.

4.3. Qualitative analysis

We now analyze qualitatively the graphs generated by each approach. Here we focus on GraphRNN models. Figure 3 (top left) shows four graphs from the Community-small dataset and four graphs from the Enzymes dataset. We then show graphs generated by variants of GraphRNNs that are trained with different node orderings (BFS, uniform, and our VI approach); these samples are representative and not cherry-picked. For Community-small, our method can capture the specific graph pattern—only one edge exists between two communities—with only one exception. The model trained with BFS orderings learns to generate two communities, but it does not generally use a single edge to connect them. The model fitted with uniform orderings fails to generate two communities. These results can be explained by the plot of adjacency matrices in Figure 3 (bottom). In this figure, we choose one graph, sample node orderings from different distributions, and plot the average of their corresponding adjacency matrices. On Community-small, the BFS order produces an adjacency matrix whose two anti-diagonal blocks are near zero. We hypothesize that this pattern across all node orderings is easier for the model to learn. The variational distribution discovers this pattern.

We perform the same analysis for the Enzymes dataset. In Figure 3 (top), the samples from the VI training method are more similar to the ground truth data than for the baseline training methods—they have the shape of long strips, and two of them contains large cycles. Figure 3 (bottom) shows the averaged adjacency matrices; we can see that the variational distribution learns to form band matrices that have most non-zeros around the diagonal. In contrast, BFS orderings scatter non-zeros to a wider range. In Appendix A.4, we provide a similar analysis for DeepGMG (which is based on graph sequences) on the Enzymes dataset.

4.4. Quality of generated graphs

Here we quantitatively assess the quality of generated graphs. Following previous works (You et al., 2018; Liao et al., 2019; Goyal et al., 2020), we measure the quality in terms of their similarity to a test set using different metrics: the degree distribution, clustering coefficients and occurrences of 4-node orbits. Then, we measure the difference

between the test set and a set of generated graphs using the maximum mean discrepancy (MMD) between their respective distributions (lower MMD indicates a better model).

Table 2 shows the MMD evaluation on the six datasets. The VI training method improves the performance of the three models in four datasets (Community-small, Enzymes, Yeast, and Cora), with some minor performance drops on the other two datasets. On Citeseer-small, the VI method exhibits a performance drop on only one metric when it is applied on GraphRNN or GraphGEN; this is somewhat consistent with our previous results that the log-likelihood improvement on this dataset is less significant. Overall, the results indicate that an autoregressive generative model trained with VI produces higher-quality graphs.

5. Conclusion

In this paper, we analyze autoregressive graph generative models that are based on either the adjacency matrix or the graph sequence. We provide an in-depth discussion of the automorphism issue that raises when calculating the marginal likelihood of the graph. Using VI, we also address the intractable marginalization over node orderings for fitting a graph generative model. The experiment results show that the variational distribution learns reasonable orderings that improve the generative model's performance. Our variational lower bound is tighter than existing bounds on the marginal log-likelihood. We evaluate models based on their test log-likelihood and find that models fitted with our VI approach exhibit better predictive performance and are able to generate higher-quality graphs than previous methods. The main limitation of our method is its scalability; thus it is not designed for large graphs. We expect future work will accelerate the algorithm to improve its scalability.

Acknowledgements

We thank Yujia Li for his insightful comments, and the anonymous reviewers for their constructive feedback. The work was supported by NSF 1850358 and NSF 1908617. Xu Han was also supported by NSF 1934553.

References

Arvind, V., Köbler, J., Rattan, G., and Verbitsky, O. Graph isomorphism, color refinement, and compactness. *computational complexity*, 26(3):627–685, 2017.

Beals, R., Chang, R., Gasarch, W., and Torán, J. On finding the number of graph automorphisms. *Chicago J. Theor. Comput. Sci*, 1999.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

Cai, D., Campbell, T., and Broderick, T. Edge-exchangeable graphs and sparsity. In *Advances in Neural Information Processing Systems*, pp. 4249–4257, 2016.

Carbonetto, P., King, M., and Hamze, F. A stochastic approximation method for inference in probabilistic graphical models. In *Advances in Neural Information Processing Systems*, 2009.

Dai, H., Nazi, A., Li, Y., Dai, B., and Schuurmans, D. Scalable deep generative modeling for sparse graphs. *arXiv preprint arXiv:2006.15502*, 2020.

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

Godsil, C. and Royle, G. F. *Algebraic graph theory*, volume 207. Springer Science & Business Media, 2001.

Goyal, N., Jain, H. V., and Ranu, S. GraphGEN: A scalable approach to domain-agnostic labeled graph generation. In *Proceedings of The Web Conference 2020*, pp. 1253–1263, 2020.

Guo, X. and Zhao, L. A systematic survey on deep generative models for graph generation. *arXiv preprint arXiv:2007.06686*, 2020.

Kim, S., Chen, J., Cheng, T., Gindulyte, A., He, J., He, S., Li, Q., Shoemaker, B. A., Thiessen, P. A., Yu, B., Zaslavsky, L., Zhang, J., and Bolton, E. E. PubChem 2019 update: improved access to chemical data. *Nucleic Acids Research*, 47(D1):D1102–D1109, 10 2018.

Klicpera, J., Bojchevski, A., and Günnemann, S. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018.

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

Liao, R., Li, Y., Song, Y., Wang, S., Hamilton, W., Duvenaud, D. K., Urtasun, R., and Zemel, R. Efficient graph generation with graph recurrent attention networks. In *Advances in Neural Information Processing Systems*, pp. 4255–4265, 2019.

Liu, C.-C., Chan, H., Luk, K., and Borealis, A. Autoregressive graph generation modeling with improved evaluation methods. In *NeurIPS'2019 Workshop on Graph Representation Learning*, 2019.

McKay, B. D. and Piperno, A. Nauty and traces user's guide (version 2.5). *Computer Science Department, Australian National University, Canberra, Australia*, 2013.

Mohamed, S., Rosca, M., Figurnov, M., and Mnih, A. Monte Carlo gradient estimation in machine learning. *arXiv preprint arXiv:1906.10652*, 2019.

Murphy, K. P. *Machine learning: a probabilistic perspective*. MIT press, 2012.

Nowicki, K. and Snijders, T. A. B. Estimation and prediction for stochastic blockstructures. *Journal of the American Statistical Association*, 96(455):1077–1087, 2001.

Paisley, J. W., Blei, D. M., and Jordan, M. I. Variational Bayesian inference with stochastic search. In *International Conference on Machine Learning*, 2012.

Ranganath, R., Gerrish, S., and Blei, D. M. Black box variational inference. In *Artificial Intelligence and Statistics*, 2014.

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

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

Shi, C., Xu, M., Zhu, Z., Zhang, W., Zhang, M., and Tang, J. Graphaf: a flow-based autoregressive model for molecular graph generation. *arXiv preprint arXiv:2001.09382*, 2020.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

Veitch, V. and Roy, D. M. The class of random graphs arising from exchangeable random measures. *arXiv preprint arXiv:1512.03099*, 2015.

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

Watts, D. J. and Strogatz, S. H. Collective dynamics of 'small-world'networks. *nature*, 393(6684):440–442, 1998.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. Simplifying graph convolutional networks. In *International conference on machine learning*, pp. 6861–6871. PMLR, 2019.

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

Yuan, H., Tang, J., Hu, X., and Ji, S. Xgnn: Towards model-level explanations of graph neural networks. *arXiv preprint arXiv:2006.02587*, 2020.

A. Appendix

A.1. Variance of the Gradient Estimators

We use the score function estimator (Williams, 1992) to obtain the gradients. In some applications, this estimator may suffer from high variance and make the training process unstable. Here, we study the variance of the score function estimator to make sure that it does not cause optimization issues in our application. To show the behavior of the optimizer, we plot the objective (the ELBO) in Figure 4(right) and the variance of the gradient estimator in Figure 4(left); both against training epochs. We can see that the objective decreases smoothly throughout optimization, indicating that the algorithm is stable. The three curves in the left plot show the variance of the gradients for different number of Monte Carlo samples; as expected, the variance decreases as the number of samples increases. Moreover, the variance from a relatively small sample size ($S = 8$) is already decently low. This is because the variational distribution $q_\phi(\pi|G)$ tends to concentrate its probability mass to a small number of node orders, which can be seen from our analysis of the variational distribution (Figure 3 and Figure 5). Considering the tradeoff between computation time and variance, we set $S = 8$ in all our experiments.

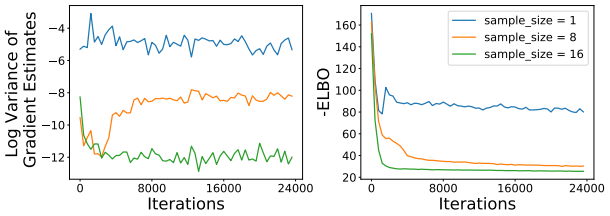

Figure 4. (Left) Training objective (ELBO) against epochs for GraphGEN on the Community-small dataset. The objective decreases smoothly throughout optimization. (Right) Log-variance of the score function gradient estimator for different number of Monte Carlo samples. Using $S = 8$ samples is enough to estimate the gradient.

A.2. Proof of Lemma 1

Lemma 1. *Let $G[V\backslash\{u\}]$ and $G[V\backslash\{v\}]$ respectively denote the subgraphs induced by $V\backslash\{u\}$ and $V\backslash\{v\}$, then u and v are in the same orbit if and only if $G[V\backslash\{u\}]$ and $G[V\backslash\{v\}]$ are isomorphic.*

Proof. Let '$\equiv$' denote the isomorphic relation. Also denote $E\backslash\{u\} = \{(i,j) \in E : i \neq u, j \neq u\}$ as the subset of edges that do not incident u.

We first show the first direction: "u and v being in the same orbit" indicates "$G[V\backslash\{u\}] \equiv G[V\backslash\{v\}]$". If u and v are in the same orbit, then $\exists f \in \text{Auto}(G) : f(v) = u$. Then $\forall i, j \in V\backslash\{u\}$, $(i,j) \in E\backslash\{u\} \iff (f(i), f(j)) \in E\backslash\{v\}$ because f is an automorphism. Then we restrict f to $V\backslash\{u\}$ and get a injection $f' : V\backslash\{u\} \to V\backslash\{v\}$, and $f'(i) = f(i)\ \forall i \in V\backslash\{u\}$. Then $\forall i, j \in V\backslash\{u\}$, $(i,j) \in E\backslash\{u\} \iff (f'(i), f'(j)) \in E\backslash\{v\}$. Therefore, f' is an isomorphism between $G[V\backslash\{u\}]$ and $G[V\backslash\{v\}]$.

We then prove by induction the second direction: "$G[V\backslash\{u\}] \equiv G[V\backslash\{v\}]$" indicates that "$u$ and v being in the same orbit" .

In the base case, we consider graphs with two nodes. Let G be $(V = \{u, v\}, E = \emptyset)$ or $(V = \{u, v\}, E = (u, v))$. In either case, we always have $G\backslash\{u\} \equiv G\backslash\{u\}$. The two nodes u and v are also in the same orbit in both cases. So the second direction holds in the base case.

Then in the induction step, we assume the second direction is true for any graph of size n, then we show that it is also true for a graph of size $n + 1$. Let $f \in \text{Auto}(G)$, and $f(u) = u'$. There are three cases: u' is v, u' is u, or u' is neither of them. If it is the first case, then we have the conclusion directly: u and v are in the same orbit.

Then we check the third case. With the same argument in the proof of the first direction, we restrict f to $V\backslash\{u\}$ and get an isomorphism: $G\backslash\{u\} \equiv G\backslash\{u'\}$. By the condition $G\backslash\{u\} \equiv G\backslash\{v\}$, we also have $G\backslash\{u'\} \equiv G\backslash\{v\}$. We then remove u from both graphs and get $G\backslash\{u, u'\} \equiv G\backslash\{u, v\}$. With the induction rule, we have that u' and v in the same orbit in $G\backslash\{u\}$. Let $g(\cdot) \in \text{Auto}(G\backslash\{u\})$ and $g(u') = v$. We extend $g(\cdot)$ to V and let $g(u) = u$, then $g \circ f$ creates an automorphism on G, and $(g \circ f)(u) = v$. Therefore, u is in the same orbit as v.

Finally, we show how to construct an $f(\cdot)$ such that $f(u) = u' \neq u$. Since $G[V\backslash\{v\}] \equiv G[V\backslash\{u\}]$, there is a isomorphism $h : V\backslash\{v\} \to V\backslash\{u\}$, and $h(u) = u'$. Note that u' cannot be u because u is not in the range of h. We extend h to the domain V and let $h(v) = u$, so h is a permutation of V. For any $i, j \in E\backslash\{v\}$, $(i,j) \in E\backslash\{v\} \iff (h(i), h(j)) \in E\backslash\{u\}$ because h is an isomorphism. It is also true that $(i,j) \in E \iff (h(i), h(j)) \in E$ because (i,j) does not incident v, and $(f(i), f(j))$ does not incident u. Since h is a permutation, the composition of h forms a group: $\{h^0, h^1, \ldots, h^K\}$. The inverse h^{-1} is the same as h^K. Let $j \in V\backslash\{v\}$, and $j = h^{-1}(i)$, $i \in V\backslash\{u\}$, then $(h(v), h(j)) = (u, i)$. With the previous argument, $(u,i) \in E \iff (h(u), h(i)) \in E$. By the compisition rule, we further have $(h(u), h(i)) \in E \iff \ldots \iff (h^K(u), h^K(i)) = (v, j) \in E$. This works for any $j \in V\backslash v$, that is, $\forall j \in V\backslash\{v\}$, $(v,j) \in E \implies (h(v), h(j)) \in E$, then h is a non-trivial automorphism on G and $h(u) \neq u'$. □

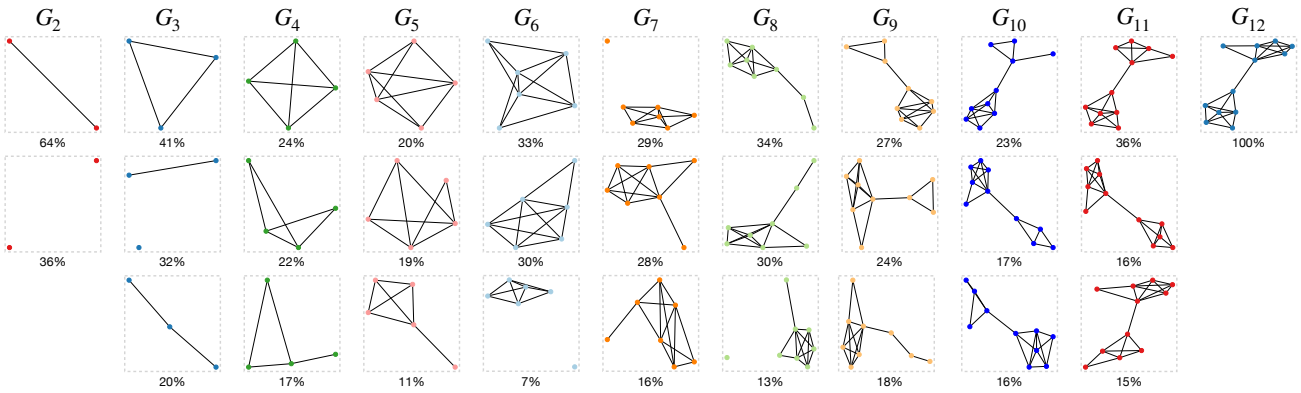

Figure 5. Graph generative sequences sampled from $q_\phi(\pi|G)$ for a graph from Community-small. The variational distribution prefers sequences of connected graphs. Similary to the distribution indicated in Figure 3 (bottom left), it first generates a comunity and then adds another one.

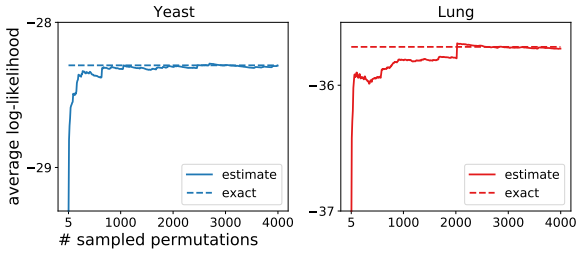

Figure 6. Comparison of estimated log-likelihood and exact log-likelihood for small graphs (fewer than 10 nodes) in the Lung and Yeast test sets. The estimation from $S = 2200$ importance samples is very accurate.

A.3. The Accuracy of the Log-Likelihood Estimation

To make sure we give an accurate estimation of the log-likelihood, we compare the estimated log-likelihood using different number of importance samples against the true log-likelihood. We compute the true log-likelihood of a graph by enumerating all possible permutations. We conduct the experiment on two datasets, Yeast and Lung. Since the calculation of the true log-likelihood is only feasible on small graphs, we keep graphs with fewer than 10 nodes in each of the two datasets. We use GraphRNN trained by variational inference as the model here, and the proposal distribution is the learned $q_\phi(\pi|G)$. Figure 6 shows the results on the two datasets. We see that when the number of samples is over 1000, the gap between the true log-likelihood and the estimated log-likelihood becomes very small (less than 0.1). When we increase the number of samples to 2200, the estimation is very accurate for both datasets. We conclude that the importance sampling estimator can be reliably used for model selection and model comparison.

A.4. Graph Sequence Pattern in DeepGMG

In Section 4, we have investigated the variational distribution when training GraphRNNs. Here we study the variational distribution when training DeepGMG. For this experiment, we also consider the Community-small and the Enzymes dataset in order to show how our model learns a set of preferred orders. We choose the smallest graph from each dataset (a graph with 12 nodes for Community-small and a graph with 10 nodes for Enzymes). For each graph, we sample 720 graph sequences from the trained variational distribution. We show the sampled graph sequences in Figure 5 and Figure 7. Without any prior knowledge, the variational distribution has strong preference for sequences of connected graphs. In addition, in Community-small, just like GraphRNN, the model prefers to generate communities one by one.

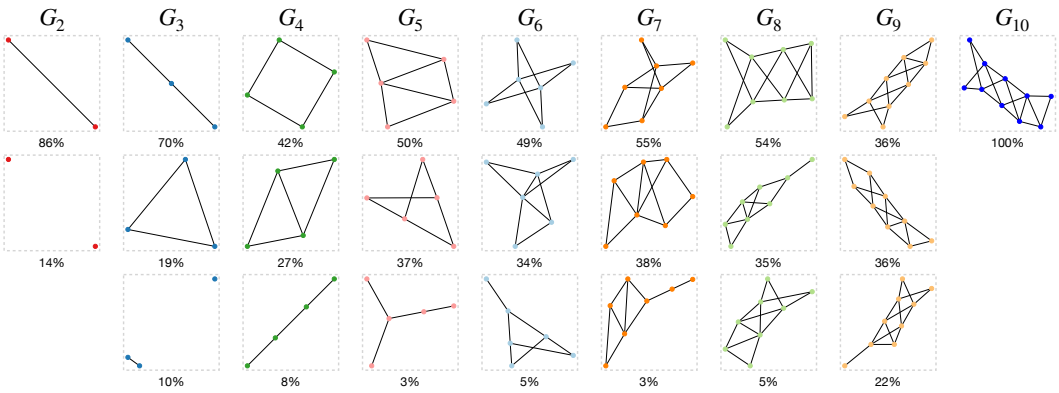

Figure 7. Graph generative sequences sampled from $q_\phi(\pi|G)$ for a graph from Enzymes. The variational distribution has a strong preference for sequences of connected graphs.