Journal of Machine Learning Research 24 (2023) 1-30 Submitted 3/22; Revised 2/23; Published 3/23

Fitting Autoregressive Graph Generative Models through Maximum Likelihood Estimation

Xu Han XU.HAN@TUFTS.EDU
Department of Computer Science
Tufts University
Medford, MA 02155, USA

Xiaohui Chen XIAOHUI.CHEN@TUFTS.EDU
Department of Computer Science
Tufts University
Medford, MA 02155, USA

Francisco J. R. Ruiz FRANRRUIZ@DEEPMIND.COM
DeepMind
5 New Street, London, UK

Li-Ping Liu LIPING.LIU@TUFTS.EDU
Department of Computer Science
Tufts University
Medford, MA 02155, USA

Editor: Mingyuan Zhou

Abstract

We consider the problem of fitting autoregressive graph generative models via maximum likelihood estimation (MLE). MLE is intractable for graph autoregressive models because the nodes in a graph can be arbitrarily reordered; thus the exact likelihood involves a sum over all possible node orders leading to the same graph. In this work, we fit the graph models by maximizing a variational bound, which is built by first deriving the joint probability over the graph and the node order of the autoregressive process. This approach avoids the need to specify ad-hoc node orders, since an inference network learns the most likely node sequences that have generated a given graph. We improve the approach by developing a graph generative model based on attention mechanisms and an inference network based on routing search. We demonstrate empirically that fitting autoregressive graph models via variational inference improves their qualitative and quantitative performance, and the improved model and inference network further boost the performance. The implementation of the proposed model is publicly available at https://github.com/tufts-ml/Graph-Generation-MLE.

Keywords: Graph generation, autoregressive graph models, variational inference

1. Introduction

Probabilistic models of graphs have been widely studied in statistics and graph theory. One of the earliest models of graphs—the Erdős-Rényi model—dates back to 1960 (Erdős and Rényi, 1960). Since then various formulations have been proposed to characterize distributions overs graphs (Watts and Strogatz, 1998; Nowicki and Snijders, 2001; Cai et al., 2016) with both data modeling and theoretical goals.

To allow for flexible and learnable distributions over graphs, recent works have extended the classic graph models by using neural networks to define such distributions. In these models, a neural network stochastically determines the connections between graph nodes. Since the network is flexible enough to capture complex correlations, the resulting model is able to learn complex graph patterns. The learned patterns are then used for inferring information about a collection of graphs or for generating new graphs with similar properties.

These models make use of popular deep learning architectures and generative models (Guo and Zhao, 2020), such as recurrent neural networks (RNNs) (You et al., 2018), generative adversarial networks (GANs) (Goodfellow et al., 2014; Wang et al., 2018) or variational autoencoders (VAEs) (Kingma and Welling, 2013; Kipf and Welling, 2016b).

A popular class of deep generative graph models are autoregressive 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), which are the focus of this paper. As opposed to many classical graph models, which typically characterize prescribed statistics of graphs, autoregressive models are designed to learn flexible graph distributions and provide easy sampling procedures. An autoregressive model generates a graph by sequentially adding nodes and edges. More concretely, the generative procedure samples an adjacency matrix by sequentially sampling its entries one row at a time. This generative process implies a certain probability distribution over graphs. In this paper, we derive this distribution in a principled manner and clarify the differences with classical graph distributions (e.g., Erdős and Rényi, 1960) and with deep graph generative models based on node representations (e.g., Kipf and Welling, 2016b).

In principle, one way to fit an autoregressive model to a given dataset is via maximum likelihood estimation (MLE). MLE maximizes the graphs' probability with respect to the model parameters. Unfortunately, the probability distribution of autoregressive models is typically intractable because there are multiple autoregressive sequences leading to the same graph (You et al., 2018; Liao et al., 2019). Evaluating the probability of a given graph would require to consider the probabilities of all such sequences, which is computationally intractable due to the large number of possible sequences.

In practice, when fitting an autoregressive graph model, some heuristics are adopted. For example, some approaches consider only a particular subset of all possible sequences. Other approaches consider a single "canonical" sequence, determined by, e.g., depth-first search (DFS) or breadth-first search (BFS) order. In this case, the training objective only considers the probability of a graph using one node order, which is a loose lower bound of the exact log-likelihood. Thus, despite the fact that these heuristics are simple and can be effective in some settings, the training objective does not correspond to rigorous MLE.

Similarly, probability-based evaluation metrics (namely, log-likelihood) are also intractable for the same reason. Instead, other evaluation metrics such as degree distribution are used, but these metrics exhibit some issues for complex graphs (Liu et al., 2019; O'Bray et al., 2021).

In this paper, we provide a method to estimate the log-likelihood for autoregressive graph models, addressing the fundamental issue of MLE described above. Besides MLE, estimating the log-likelihood of graphs brings other benefits. For example, it enables log-likelihood evaluation, standard statistical model checking and comparison, and it also opens the door for other tasks that require the log-likelihood of graph distributions, such as density-based anomaly detection.

Specifically, we sidestep the computational intractability of MLE by performing approximate posterior inference over the autoregressive sequences (more precisely, over the order of the nodes in the graph, which is equivalent). To approximate the posterior, we use variational inference (VI) (Blei et al., 2017) and maximize a lower bound of the log-likelihood. We design a neural network that infers a probability distribution over the order of the nodes. Thus, the generative model is trained with sequences that are likely to generate the observed graphs, avoiding the need to define ad-hoc orders. By doing so, this approach is able to learn graph models with significantly better performance in terms of both data fitting and graph generation than existing approaches. To measure graph generation quality, we sample graphs from the fitted model and compare their similarity to graphs from the training set; the models fitted with VI exhibit higher similarity. For evaluation metrics, we compute both log-likelihood (estimated via importance sampling) as well as maximum mean discrepancy (MMD) over some statistics (Gretton et al., 2012). The MMD based on such statistics can check whether a model learns graph properties such as small-world effects and transitivity. Models fitted with VI present significantly better evaluation metrics.

Finally, in this paper we design an improved graph generative model and inference procedure by incorporating attention mechanisms (Vaswani et al., 2017a) into the model design and routing search mechanisms (Kool et al., 2018) into the approximate posterior distribution. We demonstrate experimentally that both the new generative model and the routing-based inference network improve the quantitative and qualitative performance, as measured by the metrics described above.

Contributions. This work is an extension of our previous paper (Chen et al., 2021). We extend it in different ways: (i) we discuss the related work in more depth; (ii) we provide a thorough background section where we discuss the relation between autoregressive models and models based on exchangeable distributions of adjacency matrices; (iii) we develop a new graph generative model and a new inference network based on attention mechanisms and routing search; and (iv) we conduct additional experiments where we show the benefits of the generative model and the approximate posterior over the approach of Chen et al. (2021). Putting all together, our main contributions are:

- we provide a rigorous definition of the probability of node orders in autoregressive graph generative models;
- we analyze the relation between the calculation of graph probabilities and graph automorphism;
- we introduce VI to infer node orders and train graph models by maximizing a variational bound on the log-likelihood;
- we develop a new generative model and inference network that are based on attention mechanisms and routing search; and
- we show that fitting autoregressive graph models via VI improves their qualitative and qualitative performance, and both the attention-based generative model and the routing-based inference network improve it further.

Organization of the paper. The rest of this paper is organized as follows. In Section 2, we discuss the related work. In Section 3, we discuss two broad class of graph models (autoregressive models and exchangeable distributions) and introduce the notation of the paper. In Section 4, we consider the generation order as a random variable and analyze the relation between the graph probability and graph automorphisms. In Section 5, we introduce VI to infer the node order probability and form a suitable objective function based on a lower bound of the log-likelihood; the main ideas in Sections 4 and 5 were also included in our previous work (Chen et al., 2021). In Section 6, we incorporate attention mechanisms into the generative model and routing search into the design of the approximate posterior distribution. In Section 7, we analyze the empirical performance of graph generative models trained with VI. We conclude the paper in Section 8.

2. Related Work

The graph generation problem has been investigated for decades. Traditional methods like Erdős-Rényi random graphs (Erdős and Rényi, 1960), stochastic blockmodels (Holland et al., 1983), or the Barabási–Albert model (Albert and Barabási, 2002) usually make model assumptions like considering a particular graph family. Exponential-family random graph models (ERGMs) (Newman, 2003; Lusher et al., 2013) provide a flexible form to parameterize random graphs that have various desired properties (Snijders et al., 2006) that are typically of interest in studies of social networks, e.g., small-world effects, transitivity, or scale-free. In our work, we focus on small graphs and aim at learning their distribution without explicitly specifying their properties.

More recently, deep learning has advanced the field of generative modeling in many domains, including graph generation (Kipf and Welling, 2016b; Simonovsky and Komodakis, 2018; You et al., 2018). One branch of deep graph generative models combines a VAE and a graph neural network (GNN) to obtain a generative model based on latent representations of the nodes in the graph. One of such models is VGAE (Kipf and Welling, 2016b), which assumes that the latent node representations are *a priori* Gaussian distributed. Instead of Gaussian distributions, Mehta et al. (2019); Chen et al. (2022a) use a spike and slab prior (Griffiths and Ghahramani, 2011; Teh et al., 2007) to capture the community membership of the node representations. Such works model the node distribution and generate a graph based on the learned node characteristics.

Another branch of deep graph generative models is deep autoregressive models. These models are frequently used due to both the quality of the generated graphs and their generation efficiency and have been applied to real-world problems such as molecule modeling (Jin et al., 2018; Shi et al., 2020; Luo et al., 2021). One example of an autoregressive model is GraphRNN (You et al., 2018), which uses an RNN to model a sequential process that adds nodes and edges. In this model, the order of the nodes in the sequence is relevant, and GraphRNN assumes BFS node orders. In contrast, GRAN (Liao et al., 2019) further discusses the influence of choosing different order schemes (BFS, DFS, k-core, descending degree, etc.). These *ad-hoc* node orders can be used to form a variational lower bound on the graph log-likelihood (Liao et al., 2019). However, when the node orders are either randomly sampled from a uniform distribution or limited to a small range of canonical orders, the resulting variational bound may be loose.

One model that considers a single canonical order is GraphGEN (Goyal et al., 2020). For a given graph, GraphGEN obtains its likelihood by considering that the graph was generated according to the canonical order; 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 probability that the model assigns to that sequence.

Other research directions improve autoregressive models in different ways. For example, BiGG (Dai et al., 2020) aims at improving the model scalability by generating large sparse graphs, reducing the generation time complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}((n+m)\log n)$, where n is the number of nodes in the graph and m is the number of edges.

Another challenge of graph generative models is the qualitative evaluation of the generated graphs. One way to evaluate the quality of graph samples is to compute the distance between the graph statistics of the generated graphs and the observed graphs. The distance between two graph families can be computed via MMD (Gretton et al., 2012; You et al., 2018). The statistics of interest are typically the distribution of node statistics, e.g., node degree, graphlet counts, clustering coefficient, etc. In specific applications, other statistics may be used as well; e.g., to evaluate the synthetic molecules for drug discovery. Some useful metrics are the octanol-water partition coefficient, synthetic accessibility score, or quantitative estimation of drug-likeness. However, most of these evaluation metrics are handcrafted and may not necessarily reflect the structural properties of the graph family. Another way to evaluate the quality of generated graphs is by computing the graph likelihood, which is challenging for most generative models. In this work, we show how to estimate the likelihood of a graph using importance sampling; for that, we first formalize the definition of the graph likelihood and show that it is related to the graph automorphism problem (see also Chen et al., 2021).

3. Notation and Background

Let $G = (V, E)$ denote a graph, with V being the node set and E being the edge set. Suppose $V = \{v_1, \ldots, v_n\}$ has n nodes, and each edge in E is a two-element subset of V. If we assign the identifiers $\{1, \ldots, n\}$ to the graph nodes, then we get a *labeled graph*.[1] Each labeled graph is uniquely characterized by an adjacency matrix $\mathbf{A}$,

$$A_{i,j} = \begin{cases} 1, & \text{if } (i,j) \in E, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

In practice, the nodes in a graph G are not labeled. Thus, in what follows, we consider that G is an *unlabeled graph*. Unlike labeled graphs, there is not one-to-one correspondence between adjacency matrices and unlabeled graphs. Instead, there is a set $\mathcal{A}(G)$ of distinct adjacency matrices derived from G, such that each $\mathbf{A} \in \mathcal{A}(G)$ corresponds to the adjacency matrix of a labeled graph obtained by assigning identifiers (integer labels) to the nodes of G. The number of distinct adjacency matrices in $\mathcal{A}(G)$ is

$$|\mathcal{A}(G)| = \frac{n!}{|\mathit{\Gamma}(G)|}, \tag{2}$$

where $\mathit{\Gamma}(G)$ denotes the set of automorphisms of the graph G (Harary et al., 1973).

1. The phrase "labeled graph," which is often used in mathematics, means that graph nodes are labeled with distinct integers. It must not be confused with the "labels" in the context of supervised learning.

In this work, we consider a distribution $p(G)$ over $\mathcal{G}$, where $\mathcal{G}$ denotes all finite graphs. Typically, we define $p(G)$ by considering a distribution $p(\mathbf{A})$ over adjacency matrices in $\mathcal{A}$, where $\mathcal{A} = \cup_{G \in \mathcal{G}} \mathcal{A}(G)$ denotes the set of all adjacency matrices derived from finite graphs. From the definition of unlabeled graphs, we have that $p(G)$ can be obtained from $p(\mathbf{A})$ as

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

To define the distribution $p(\mathbf{A})$, one approach is to use a neural network that probabilistically generates each row of $\mathbf{A}$ in a sequential manner. We discuss these autoregressive methods in Section 4 and design a new generative model in Section 6.

Unfortunately, the computation of Eq. 3 requires to enumerate all adjacency matrices in $\mathcal{A}(G)$. This enumeration is computationally expensive, because it is equivalent to enumerating all labeled graphs derived from G, which is a non-trivial problem (Harary et al., 1973). We address this issue in Sections 5 and 6.

Two types of generative models. Before discussing autoregressive models, we relate the probability distribution above with classic graph distributions. Most classic graph distributions consider labeled graphs (Frieze and Karoński, 2015, Chapter 1). Given the one-to-one relationship between labeled graphs and adjacency matrices, such distributions are essentially distributions over adjacency matrices. For example, the Erdős-Rényi model (Erdős and Rényi, 1960) defines a distribution $p(\mathbf{A})$ through independent Bernoulli distributions over the entries of $\mathbf{A}$.

When we use a classical distribution $p(\mathbf{A})$ to model a graph (often a network), we do not wish the probability calculations to be affected by the specific choice of node labels. To satisfy this requirement, we need $p(\mathbf{A})$ to be an *exchangeable* distribution over $\mathcal{A}$ (Orbanz and Roy, 2014)—if $p(\mathbf{A})$ is exchangeable, then it satisfies $p(\mathbf{A}) = p(\mathbf{A}_\pi)$, where $\mathbf{A}_\pi$ denotes a row and column permutation of $\mathbf{A}$ specified by a permutation π. In order to make a distribution $p(\mathbf{A})$ exchangeable, the model often needs to know the number of nodes beforehand, therefore, $p(\mathbf{A})$ is often obtained by conditioning on n as $p(\mathbf{A}) = \sum_{n=1}^{\infty} p(\mathbf{A} \mid n) p(n)$.

Some more modern graph models, such as the VGAE (Kipf and Welling, 2016b) and alike, define a set of node representations and then generate an adjacency matrix $\mathbf{A}$. These models also consider exchangeable distributions $p(\mathbf{A})$. Because of the property mentioned above, these methods are referred to as "parallel generation" (Lippe and Gavves, 2020; Chen et al., 2022b) or "generation with fixed-set of nodes" (You et al., 2018). These descriptions of $p(\mathbf{A})$ are rooted in the exchangeability property of $p(\mathbf{A})$.

In this work, the graph distribution $p(G)$ is defined by Eq. 3 and does not require the underlying distribution $p(\mathbf{A})$ to be exchangeable, because Eq. 3 obtains the sum over all possible combinations of node labels. For the same reason, if a graph distribution $p(G)$ is defined through a non-exchangeable distribution $p(\mathbf{A})$, then $p(G)$ should be defined by Eq. 3; otherwise it is not possible to determine which adjacency matrix $\mathbf{A} \in \mathcal{A}(G)$ to consider. By removing the exchangeability requirement, the distribution $p(G)$ from Eq. 3 allows for flexible distributions to model graph structures, making it appropriate for graph-level tasks. However, it also presents computational challenges, as it inevitably needs to consider all adjacency matrices derived from the graph G. We address this difficulty in this work.

Algorithm 1 Autoregressive generation of adjacency matrices

```
Require: The maximum size of the matrix, $n_{\max} > 0$
  $i \leftarrow 1$
  while $i \leq n_{\max}$ do
      $i = i + 1$
      for $j = 1$ to $i$ do
          sample $A_{i,j}$
      end for
      sample stop variable $S \in \{0, 1\}$
      if stop then
          break
      end if
  end while
```

4. Autoregressive Graph Generation

In this section we describe the autoregressive generative model $p(\mathbf{A})$ of (non-exchangeable) adjacency matrices. An autoregressive model considers that the entries of the adjacency matrix $\mathbf{A}$ are generated sequentially (You et al., 2018). This sequential sampling procedure defines the distribution $p(\mathbf{A})$.

More specifically, the autoregressive model considers the lower triangular matrix $\mathbf{L}$ such that the adjacency matrix $\mathbf{A} = \mathbf{L} + \mathbf{L}^\top$ (the adjacency matrix is symmetric for undirected graphs). The model generates the matrix $\mathbf{L}$ by sequentially sampling each of its rows. After a row is generated, it decides whether to stop ($S = 1$) or not to stop ($S = 0$). Since each $\mathbf{L}$ uniquely determines $\mathbf{A}$ and vice-versa; we have that $p(\mathbf{A}) = p(\mathbf{L})$, and

$$p(\mathbf{A}) = p(S = 1 \mid \mathbf{L}) \prod_{t=2}^{n} p(S = 0 \mid \mathbf{L}_{1:(t-1)}) p(L_{t,:} \mid \mathbf{L}_{1:(t-1)}), \tag{4}$$

where $\mathbf{L}_{1:(t-1)}$ denotes the submatrix formed from the first $t-1$ rows and columns of $\mathbf{L}$, and $L_{t,:}$ is the t-th row of $\mathbf{L}$. Note that $L_{1,:}$ is empty so the product starts from $t = 2$. The probability $p(S = 0 \mid \mathbf{L}_{1:(t-1)})$ corresponds to the case of continuing the generation after $t-1$ rows, while $p(S = 1 \mid \mathbf{L})$ corresponds to the case of stopping generation after n rows. This generation procedure is described in Algorithm 1.

Many models in the literature use this autoregressive approach to generate $\mathbf{A}$. For example, the formulation by Liao et al. (2019), which generates graph nodes in batches, can be expressed as an autoregressive model of the form above. GraphGEN (Goyal et al., 2020; Dai et al., 2020), which generates the sparsity pattern of each row of $\mathbf{L}$, is also in this form. In Section 6.2, we develop a new autoregressive model for $p(\mathbf{A})$.

The autoregressive generation procedure assumes that nodes are generated sequentially, following some *node order* or *node permutation* π. There are $n!$ possible permutations, and each permutation π assigns an identifier or label to all nodes. Thus, every node order π determines a labeled graph, which means that the pair (G, π) uniquely determines a particular adjacency matrix $\mathbf{A}$ from the set $\mathcal{A}(G)$; we denote this as $(G, \pi) \rightarrow \mathbf{A}$. The converse is not true: we cannot uniquely identify π from $\mathbf{A}$. Indeed, a matrix $\mathbf{A}$ corresponds

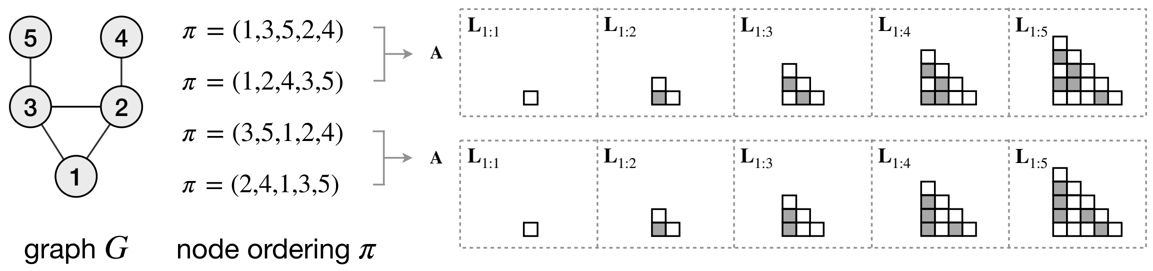


Figure 1: The relationship between the node order π and the lower triangular matrix $\mathbf{L}$ of an adjacency matrix $\mathbf{A}$. Given a graph G (left), several node orders π (middle) specify the same matrix $\mathbf{L}$, so we cannot uniquely identify π from a given $\mathbf{L}$.

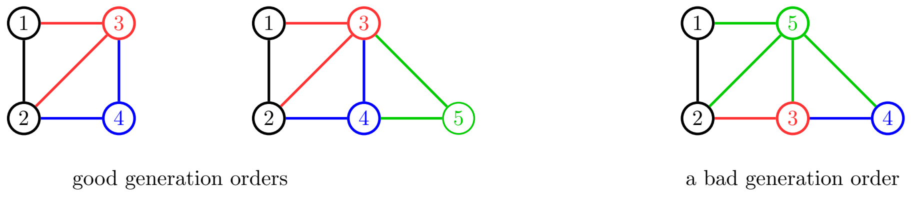


Figure 2: Generation patterns from different node orders. The two left-most graphs share the same generating pattern: each node is connected to its two preceding nodes. The third graph is the same as the second one, but the generating pattern is harder to describe because of a different node order.

to multiple node orders if G has multiple automorphisms. We illustrate this through an example in Figure 1. The first two node orders ($\pi = (1, 3, 5, 2, 4)$ and $\pi = (1, 2, 4, 3, 5)$) determine $\mathbf{A}$, but we cannot uniquely identify one of them from $\mathbf{A}$ (in particular, we cannot distinguish the node pairs $(2, 4)$ and $(3, 5)$). Similarly, we cannot identify a unique node order π from $\mathbf{A}'$.

Remark. The generation order plays an important role when fitting autoregressive generative models, because the probability $p(\mathbf{A})$ may be significantly different for two different adjacency matrices even if they correspond to the same graph. Ideally, the conditional probability $p(L_{t,:} \mid \mathbf{L}_{1:(t-1)})$ of the autoregressive model needs to capture the patterns in the data. Therefore, a good node order should form generating steps that share common generation patterns, so that it is easier for the autoregressive model to learn such patterns. Figure 2 illustrates this idea through an example. Consider the following generation pattern: "connecting each new node to its two preceding nodes." The first two graphs in the figure have been assigned a node order that follows this pattern. The third graph is the same as the second one, but the generating pattern is more complex because it uses a different node order. Therefore, it is important to identify a good generation order for the generative model to model graphs in a dataset.

Now we consider the calculation of the graph probability $p(G)$ in Eq. 3 with the underlying distribution $p(\mathbf{A})$ defined by an autoregressive model. As discussed in Section 3, this requires to consider all adjacency matrices in the set $\mathcal{A}(G)$. However, not only the summation itself is hard, but enumerating all these adjacency matrices is also a complex problem.

To address the latter issue, we consider the summation over node orders π instead of adjacency matrices $\mathbf{A}$. To do so, we consider that π is a random variable and formally specify a joint distribution $p(G, \pi)$, such that we recover the distribution $p(G)$ by marginalization,

$$p(G) = \sum_{\pi} p(G, \pi). \tag{5}$$

Obtaining $p(G)$ from Eq. 5 is easier than from Eq. 3 because the marginalization space is easier to characterize. Although Eq. 5 remains intractable due to the large number of terms in the sum, in Section 5 we derive a variational bound on $p(G)$.

We next show how the joint $p(G, \pi)$ from Eq. 5 relates to the probability $p(\mathbf{A})$ from Eq. 4. As discussed above, the pair (G, π) uniquely determines $\mathbf{A}$, but the converse is not true: given $\mathbf{A}$, there may be multiple consistent node orders π (see Figure 1). Let $\mathit{\Pi}(\mathbf{A})$ be the set of all possible node orders that give the same adjacency $\mathbf{A}$, i.e.,

$$\mathit{\Pi}(\mathbf{A}) = \{\pi : A_{\pi_i, \pi_j} = \mathbb{1}[(\pi_i, \pi_j) \in E], \forall i, j \in V\}, \tag{6}$$

where $\mathbb{1}[\cdot]$ is the indicator function, which takes value 1 or 0 depending on whether its argument is true or false. If treated as a mapping of the node set, each $\pi \in \mathit{\Pi}(\mathbf{A})$ is an automorphism of the graph.[2] Thus, the set $\mathit{\Pi}(\mathbf{A})$ contains all automorphisms of G because any automorphic mapping of the graph keeps the adjacency matrix unchanged (Harary et al., 1973, Chapter 1). Therefore, $|\mathit{\Pi}(\mathbf{A})|$ is the number of automorphisms of the graph defined by $\mathbf{A}$.

We define the conditional distribution $p(\pi|\mathbf{A})$ as a uniform,

$$p(\pi|\mathbf{A}) := \begin{cases} \frac{1}{|\mathit{\Pi}(\mathbf{A})|}, & \text{if } \pi \in \mathit{\Pi}(\mathbf{A}). \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

Then, the joint $p(G, \pi)$ in Eq. 5 is

$$p(G, \pi) = \frac{1}{|\mathit{\Pi}(\mathbf{A})|} p(\mathbf{A}). \tag{8}$$

The number of automorphisms $|\mathit{\Pi}(\mathbf{A})|$ is a constant for any given graph, so we do not need to compute this quantity for model training or model comparison. If we need to evaluate Eq. 8, there are approximate methods for computing $|\mathit{\Pi}(\mathbf{A})|$. Note that the computation of $|\mathit{\Pi}(\mathbf{A})|$ is a well-studied classic problem in graph theory. The time complexity of computing $|\mathit{\Pi}(\mathbf{A})|$ is $\exp\left(\mathcal{O}(\sqrt{n \log n})\right)$ (Beals et al., 1999). The Nauty package (McKay and Piperno, 2013) uses various heuristics and can efficiently find this number for most graphs. In practice, it can compute $|\mathit{\Pi}(\mathbf{A})|$ for a graph with thousands of nodes in less than 10^{-3} seconds.

Can we avoid the marginalization? Some previous works attempt to avoid the marginalization in Eq. 5 by using a single generation order. One example is GraphGEN (Goyal et al.,

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

2020), which uses a canonical order $\pi^\star$ to identify a single adjacency matrix $\mathbf{A}^\star$ and thus avoid the summation in Eq. 5 (or Eq. 3). Although this strategy is appealing, we show here that it brings other issues. The canonical order $\pi^\star$ and the corresponding adjacency matrix $\mathbf{A}^\star$ imply that the distribution $p(G)$ is defined as $p(G) = p(\mathbf{A}^\star)$. This requires $p(\mathbf{A}) = 0$ for any $\mathbf{A} \in \mathcal{A}(G)$ such that $\mathbf{A} \neq \mathbf{A}^\star$, according to Eq. 3.

This strategy presents two issues. First, finding a strict canonical order is a non-trivial task—the difficulty is the same as the isomorphism test. In addition, the canonical order $\pi^\star$ may not be the best generation order: algorithms for computing canonical orders are usually not designed to discover common construction patterns (see the discussion of Figure 2).

Second, it is hard to guarantee that the generative procedure follows the canonical order. In fact, there is not a straightforward way to control the generation order because the canonical order is computed retrospectively after the graph G is generated. Due to this difficulty, GraphGEN does not guarantee that samples from the model follow the canonical order. In other words, a sample from GraphGEN may be generated with a node order 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 fitted the GraphGEN model to the Community-small dataset and then sampled graphs from the fitted model. We found that only 9.1% of the generated graphs follow the canonical order.

5. A Variational Bound and an Approximation of the Log-likelihood

Here we present a method to train an autoregressive graph generative model without having to pre-specify the node order. We achieve that by maximizing a variational lower bound of the log marginal likelihood.

In particular, consider a generative model $p_\theta(\mathbf{A}_\pi)$ parameterized by θ. We obtain the marginal likelihood $p_\theta(G)$ by marginalizing out the node order π from the joint $p_\theta(G, \pi)$ (Eq. 8). Since the marginalization is intractable, we introduce a *variational distribution* $q_\phi(\pi \mid G)$, parameterized by ϕ, that approximates the posterior $p_\theta(\pi \mid G)$, and we form the variational lower bound $L(\theta, \phi, G) \leq \log p_\theta(G)$ (Blei et al., 2017), given by

$$L(\theta, \phi, G) = \mathbb{E}_{q_\phi(\pi \mid G)} \left[\log p_\theta(G, \pi) - \log q_\phi(\pi \mid G)\right]. \tag{9}$$

Variational inference maximizes Eq. 9 with respect to both the model parameters θ and the variational parameters ϕ. We discuss the optimization algorithm in Section 5.1 and the form of the variational distribution $q_\phi(\pi \mid G)$ in Section 6.1.

5.1 Maximizing the variational lower bound

To maximize the lower bound $L(\theta, \phi, G)$ in Eq. 9, we need its gradients with respect to both θ and ϕ, which are intractable. We approximate the gradient with respect to θ via Monte Carlo estimation. And a basic method of calculating the gradient with respect to ϕ is 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 $\pi^{(s)} \sim q_\phi(\pi \mid G)$ for

Algorithm 2 VI algorithm for training a graph model based on the adjacency matrix $\mathbf{A}$

```
Input: Dataset of graphs $\mathbb{G} = \{G_1, \dots, G_n\}$, model $p_\theta$, variational distribution $q_\phi$
Output: Learned parameters $\theta$ and $\phi$
repeat
    for $G \in \mathbb{G}$ do
        Sample $\pi^{(1)}, \dots, \pi^{(S)} \overset{\text{iid}}{\sim} q_\phi(\pi \mid G)$
        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 $\nabla_\theta \leftarrow \nabla_\theta L(\theta, \phi, G)$ by Eq. 10
        Compute $\nabla_\phi \leftarrow \nabla_\phi L(\theta, \phi, G)$ by Eq. 12
        Update $\phi$, $\theta$ using the gradients $\nabla_\phi$, $\nabla_\theta$
    end for
until convergence of the parameters $(\theta,\ \phi)$
```

$s = 1, \dots, S$, yielding

$$\nabla_\theta L(\theta, \phi, G) \approx \frac{1}{S} \sum_{s=1}^{S} \nabla_\theta \log p_\theta(G, \pi^{(s)}), \tag{10}$$

$$\nabla_\phi L(\theta, \phi, G) \approx \frac{1}{S} \sum_{s=1}^{S} \Big[\log p_\theta(G, \pi^{(s)}) - \log q_\phi(\pi^{(s)} \mid G) \Big] \nabla_\phi \log q_\phi(\pi^{(s)} \mid G). \tag{11}$$

Eq. 10 shows that the parameters θ of the model are optimized under node orders π sampled from the approximate posterior $q_\phi(\pi^{(s)} \mid G)$. That is, fitting the model does not require to define ad-hoc node orders π; rather, the (approximately) most likely node orders are used.

To reduce the variance of the gradient estimator with respect to ϕ, we use the leave-one-out estimator (Salimans and Knowles, 2014; Kool et al., 2019; Richter et al., 2020) instead of the vanilla score function estimator from Eq. 11. This estimator is given by:

$$\nabla_\phi L(\theta, \phi, G) \approx \frac{1}{S-1} \sum_{s=1}^{S} \left[\log \frac{p_\theta(G, \pi^{(s)})}{q_\phi(\pi^{(s)} \mid G)} - \frac{1}{S} \sum_{j=1}^{S} \log \frac{p_\theta(G, \pi^{(j)})}{q_\phi(\pi^{(j)} \mid G)} \right] \nabla_\phi \log q_\phi(\pi^{(s)} \mid G). \tag{12}$$

A model trained with uniformly distributed random node orders can be seen as using a uniform variational distribution $q_\phi(\pi^{(s)} \mid G)$ (see Liao et al., 2019). However, this approach yields a loose bound when the posterior distribution $p_\theta(\pi \mid G)$ is far from uniform.

We summarize the VI training procedure in Algorithm 2. The algorithm can be applied to any autoregressive model that has a computable $p(\mathbf{A})$, such as GraphRNN or GraphGEN.

5.2 Approximating the log-likelihood through importance sampling

The approximate posterior $q_\phi(\pi \mid G)$ also allows us to approximate the log marginal likelihood of a fitted model. In particular, we approximate $\log p_\theta(G)$ using importance sampling (Murphy, 2012; Owen, 2013), with $q_\phi(\pi \mid G)$ as the proposal distribution:

$$\log p_\theta(G) \simeq \log \Big(\frac{1}{L} \sum_{s=1}^{L} \frac{p_\theta(G, \pi^{(s)})}{q_\phi(\pi^{(s)} \mid G)} \Big), \tag{13}$$

where $\pi^{(s)} \sim q_\phi(\pi \mid G)$ for $s = 1, \ldots, L$. The importance sampling approximation uses L samples, where L is typically higher than the number of samples S used for training in Section 5.1. For any finite value of L, Eq. 13 is a biased estimator because its expectation is still a lower bound of the true log marginal likelihood, but the lower bound monotonically approaches $\log p_\theta(G)$ as L grows. In our experiments, we found that $L = 1{,}000$ gives an accurate estimation (see Figure 5).

6. A New Inference Network and Generative Model

In this section, we first develop a routing-based network architecture to construct an efficient variational distribution $q_\phi(\pi \mid G)$. Then we propose a new generative model $p_\theta(G, \pi)$ that uses an attention mechanism to improve the model's flexibility.

6.1 Routing-based inference network

The main goal of the inference network in this subsection is to speed up the inference method of Chen et al. (2021). We first note that this is a *dynamic ordering* problem, since a node in the sequence determines how later nodes are (probabilistically) compared. For example, the first node in a BFS-ordered sequence affects the order of later nodes. Therefore, algorithms that predict a static ordering are not applicable here; this includes ranking algorithms (Liu et al., 2009), which often predict the ordering of items based on their scores or pairwise comparisons. Note that the (stochastic) orders predicted by a ranking algorithm are considered static in this context because the (probabilistic) pair-wise comparison of two nodes is not affected by the order of other nodes. Secondly, we note that Algorithm 2 requires to explicitly evaluate $q_\phi(\pi \mid G)$; this prevents us from using neural models that compute probabilistic permutation matrices (e.g., Mena et al., 2018; Paulus et al., 2020), since the distribution is not available in closed form.

Instead, we opt for neural models designed for the travelling salesperson problem (TSP) (Kool et al., 2018; Joshi et al., 2019, 2020), which can *dynamically* order nodes. Unlike TSPs, which typically consider fully connected weighted graphs, we consider sparse graphs, and we leverage this sparsity to save computation. Thus, in this subsection, we adapt the attention model of Kool et al. (2018) to our problem to balance flexibility and efficiency.

As discussed in Section 5, the variational distribution $q_\phi(\pi \mid G)$ approximates the intractable posterior $p_\theta(\pi \mid G)$. A convenient choice for $q_\phi(\pi \mid G)$ is to express it as sequence,

$$q_\phi(\pi \mid G) = q_\phi(\pi_1 \mid G) \prod_{t=2}^{n} q_\phi(\pi_t \mid G, \pi_{1:(t-1)}). \tag{14}$$

Here, the term $q_\phi(\pi_t \mid G, \pi_{1:(t-1)})$ determines the probability over the next node in the sequence π_t, conditioned on both the graph structure and the partial order $\pi_{1:(t-1)}$ already formed from previous steps. Our previous work (Chen et al., 2021) uses a multi-layer GNN to define this conditional probability. Thus, to draw a single sample π from $q_\phi(\pi \mid G)$, the GNN needs to be applied n times—once for each term in Eq. 14. Therefore, the approach of Chen et al. (2021) may suffer from high computational complexity.

To obtain a more efficient procedure for sampling from $q_\phi(\pi \mid G)$, we first note that the problem of sampling node orders is similar to the traveling salesman problem and the

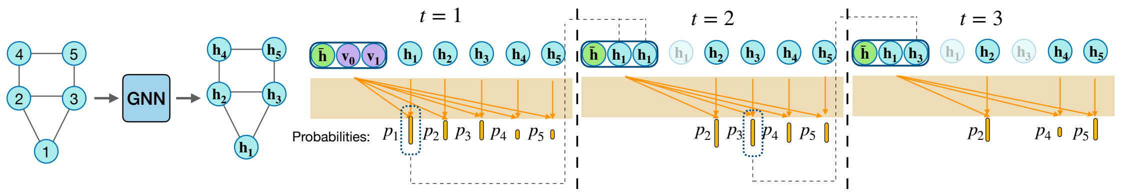


Figure 3: An overview of the node order inference model based on routing search (Rout-VI). A graph neural network captures the graph topological information and produces a representation vector for each node. At each step t, the selection probability for each node is determined by performing attention between the node representation and the context vector. The context vector at step $t+1$ is updated based on the node selection in step t.

routing problem, for which Kool et al. (2018) develop a neural network model based on attention mechanisms. Using their idea of routing search, we propose a new architecture to build the variational distribution $q_\phi(\pi \mid G)$ that requires one single application of the GNN. We call this approach Rout-VI.

Rout-VI is represented in Figure 3. The input to the GNN includes the graph structure G and some initial node representations $\mathbf{h}^0 = [\mathbf{h}_1^0, \ldots, \mathbf{h}_n^0]$, which are initialized as learnable vectors (alternatively, they could be initialized using observed node features, if available). That is, the GNN computes

$$(\mathbf{h}_1, \ldots, \mathbf{h}_n) = \text{GNN}_\phi(G, (\mathbf{h}_1^0, \ldots, \mathbf{h}_n^0)). \tag{15}$$

The GNN learns the node representations by recursively aggregating and updating each node representation from its neighboring nodes, i.e.,

$$\mathbf{h}_i^{l+1} = \text{UPDATE}\left(\mathbf{h}_i^l, \text{AGG}\left(\left\{\mathbf{h}_j^l : (i,j) \in E\right\}\right)\right), \tag{16}$$

where UPDATE$(\cdot)$ denotes an update function and AGG$(\cdot)$ denotes an aggregation function. Wu et al. (2020) survey possible instantiations of the two functions, and we omit the details here. We use the graph convolutional network (GCN) (Kipf and Welling, 2016a) in our work, and we set the dimension of the node representations to d.

Rout-VI uses the node representations obtained from the GNN to sample the node order π. For that, it uses a *context vector* $\mathbf{c}_t$ that represents the context at step t. We define the context vector as

$$\mathbf{c}_t = \begin{cases} [\bar{\mathbf{h}}, \mathbf{v}_0, \mathbf{v}_1], & \text{if } t = 1, \\ [\bar{\mathbf{h}}, \mathbf{h}_{\pi_{t-1}}, \mathbf{h}_{\pi_1}], & \text{if } t > 1. \end{cases} \tag{17}$$

Here, $[\cdot,\cdot,\cdot]$ denotes the concatenation of the three input vectors. The vectors $\mathbf{h}_{\pi_{t-1}}$ and $\mathbf{h}_{\pi_1}$ are obtained from the GNN as described above; Rout-VI uses the representations of the nodes sampled in the previous step (π_{t-1}) and in the first step (π_1). The vector $\bar{\mathbf{h}}$ is a global representation that encodes information about the entire graph by using a pooling function $\bar{\mathbf{h}} = \text{POOL}(\mathbf{h}_1, \ldots, \mathbf{h}_n)$; we use the average of all node representations. The parameters $\mathbf{v}_0$ and $\mathbf{v}_1$ are learnable vectors that are used only when sampling π_1 in the first time step.

Rout-VI uses the context vectors $\mathbf{c}_t$ to obtain the sampling probabilities of node orders. In particular, it obtains the logits $\{\ell_k^t\}$ of the distribution $q_\phi(\pi_t \mid G, \pi_{1:(t-1)})$ by comparing the context vector $\mathbf{c}_t$ and the node representations,

$$\ell_k^t = \frac{\mathbf{c}_t^\top \mathbf{W}^{\mathrm{q}} \mathbf{W}^{\mathrm{k}} \mathbf{h}_k}{\sqrt{d}}, \quad k \notin \pi_{1:(t-1)}. \tag{18}$$

This resembles the attention calculation, where $\mathbf{W}^{\mathrm{q}}$ and $\mathbf{W}^{\mathrm{k}}$ are learnable matrices. Given the logits, Rout-VI obtains the probabilities as

$$q_\phi(\pi_t \mid G, \pi_{1:(t-1)}) = \frac{\exp\{\ell_{\pi_t}^t\}}{\sum_{k \notin \pi_{1:(t-1)}} \exp\{\ell_k^t\}}, \quad \pi_t \notin \pi_{1:(t-1)}, \tag{19}$$

where the restriction $\pi_t \notin \pi_{1:(t-1)}$ ensures that nodes that have been previously sampled cannot be sampled again. With this architecture, the sampling of π_t at step t takes into account the overall graph structure (through $\bar{\mathbf{h}}$) and the information from the previous node (through $\mathbf{h}_{\pi_{t-1}}$). The information about the first node ($\mathbf{h}_{\pi_1}$) is useful for graphs with cycles.

We emphasize that the variational distribution can be used with any generative model that has an explicit $p(\mathbf{A})$ (this includes, but is not limited to, the attention-based generative model to be developed below). Note that different node orders π corresponding to the same graph automorphism (in terms of both the structure $\mathbf{A}$ and possible node features) have the same probability under the variational distribution. This is a desired property because these node orders have the same probability $p(G, \pi)$ according to Eqs. 7 and 8 and thus the same probability under the true posterior.

6.2 Attention-based generative model

We now design an autoregressive deep generative model that is based on attention mechanisms. We call it double-attention graph generative model (DAGG). Compared with recent works, the key innovation of the proposed design is to use attention mechanisms in computing connections and updating node representations in the generation process. It aims at achieving a good balance between generation efficiency and expressiveness of node representations.

To construct the new model, we devise a neural network to implement the two probabilities $p(S \mid \mathbf{L}_{1:(t-1)})$ and $p(L_{t,:} \mid \mathbf{L}_{1:(t-1)})$ in Eq. 4 (recall that $\mathbf{L}$ denotes the lower triangular part of $\mathbf{A}$, following the notation from Section 4).

Suppose that G_{t-1} is the partially generated graph at time t, and each node in it has a representation vector, and all their vectors are denoted by $\mathbf{H}_t = (\mathbf{h}_1^t, \ldots, \mathbf{h}_{t-1}^t)$. In the first step, $\mathbf{H}_t$ is empty.

DAGG first uses attention to compute a new representation vector $\mathbf{h}_t^t$ a the t-th step:

$$\mathbf{r}_t^t = \mathrm{MHA}((\mathbf{h}^*)^\top, \mathbf{H}_t, \mathbf{H}_t), \tag{20}$$
$$\mathbf{h}_t^t = \mathrm{LINEAR}(\mathbf{r}_t^t). \tag{21}$$

Here, $\mathbf{h}^*$ is a global learnable vector and the notation LINEAR$(\cdot)$ represents a linear layer. The multi-head attention $\mathbf{r} = \mathrm{MHA}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$ (Vaswani et al., 2017a) is formally defined as

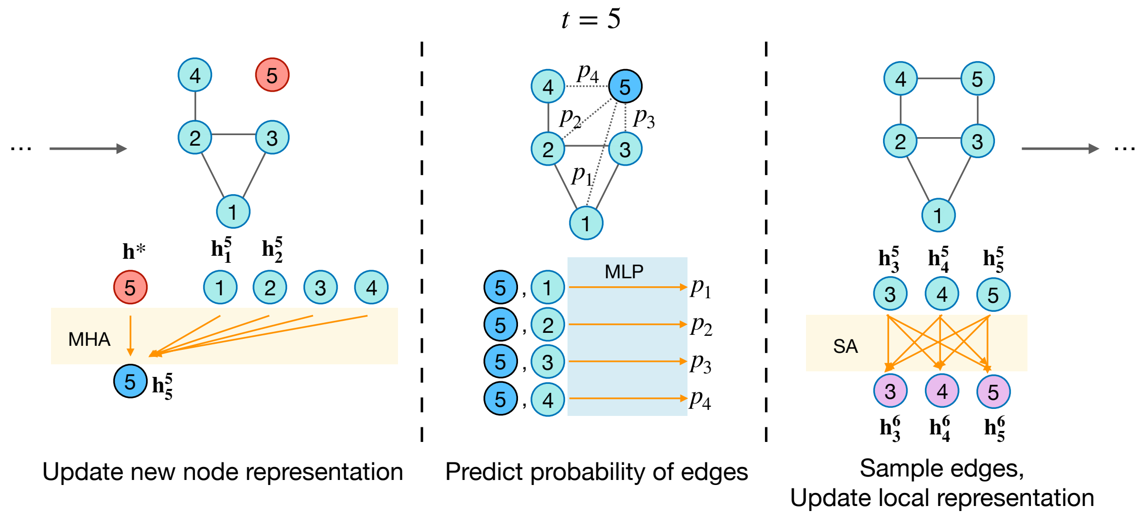

$t = 5$
MHA
MLP
SA
$\mathbf{h}^*$
$\mathbf{h}_1^5$ $\mathbf{h}_2^5$
$\mathbf{h}_5^5$
p_1 p_2 p_3 p_4
$\mathbf{h}_3^5$ $\mathbf{h}_4^5$ $\mathbf{h}_5^5$
$\mathbf{h}_3^6$ $\mathbf{h}_4^6$ $\mathbf{h}_5^6$
Update new node representation
Predict probability of edges
Sample edges, Update local representation

Figure 4: An overview of the generation process of the DAGG model at step $t = 5$. The node numbers represent a particular node order $\pi_{1:5}$. **(Left)** The representation of the newly added node 5 is given by MHA and depends on the current partial graph. **(Middle)** The probability for connections between node 5 and the other nodes are obtained from the current node representations. Here, connections to node 3 and node 4 are sampled. **(Right)** After having sampled the new connections, the node representations of nodes 3, 4, and 5 are updated by applying SA to these nodes.

follows:

$$\mathbf{s}_a = \mathrm{softmax}\left(\frac{(\mathbf{Q}\mathbf{W}_a^q)\cdot(\mathbf{K}\mathbf{W}_a^k)^\top}{\sqrt{d}}\right)\cdot(\mathbf{V}\mathbf{W}_a^v), \quad \text{for } a = 1,\dots,H, \tag{22}$$

$$\mathbf{r} = \mathrm{CONCAT}(\mathbf{s}_1,\dots,\mathbf{s}_H). \tag{23}$$

Here, d is the number of columns in $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$; and H is the number of attention heads, each of which has parameters $\mathbf{W}_a^q$, $\mathbf{W}_a^k$, and $\mathbf{W}_a^v$. Vectors from different attention heads are concatenated to get the output $\mathbf{r}$.

In summary, we use a global vector $\mathbf{h}^*$ to query previous node representations and compute a representation vector $\mathbf{h}_t^t$ for the new node.

Then, DAGG decides whether to stop at this step based on the representation vector $\mathbf{h}_t^t$:

$$p(S = 1 \mid \mathbf{L}_{1:(t-1)}) = \mathrm{sigmoid}(\mathrm{MLP}(\mathbf{h}_t^t)), \tag{24}$$

where the notation MLP$(\cdot)$ denotes a multi-layer perceptron, and $\mathrm{sigmoid}(x) = \frac{1}{1+e^{-x}}$. If the model decides to proceed with a new node ($S = 0$), then it uses $\mathbf{h}_t^t$ as the representation of that node. It then decides connections to the new node and updates the relevant representations. In particular, it samples $L_{t:} = (L_{t,1}, \dots L_{t,t-1})$, i.e., the edges between the new node and previously generated nodes with the following probabilities:

$$p(L_{t,\tau} = 1 | \mathbf{L}_{1:(t-1)}) = \text{sigmoid}(\text{MLP}([\mathbf{h}_\tau^t, \mathbf{h}_t^t])), \quad \tau = 1, \ldots, t-1, \tag{25}$$

where an MLP is applied to the vector formed by concatenating the representations $\mathbf{h}_t^t$ and $\mathbf{h}_{t'}^t$ of the two nodes involved.

DAGG then updates the node representations of the new node and the nodes connected to it, as determined by $L_{t:}$. For that, we use self-attention. Let $\mathbf{H}'_t$ be the matrix formed by the representation $\mathbf{h}_t^t$, as well as the representations $\mathbf{h}_\tau^t$ of the nodes τ that are connected to the new node, i.e., $\mathbf{H}'_t = (\mathbf{h}_\tau^t : L_{t,\tau} = 1 \text{ or } t = \tau)$. We update their representations as

$$(\mathbf{h}_\tau^{t+1} : L_{t,\tau} = 1 \text{ or } t = \tau) = \text{LINEAR}(\text{SA}(\mathbf{H}'_t)), \tag{26}$$

where $\text{SA}(\mathbf{H}'_t) = \text{MHA}(\mathbf{H}'_t, \mathbf{H}'_t, \mathbf{H}'_t)$ denotes the self-attention function (Vaswani et al., 2017b). The representations of nodes not connected to t remain unchanged, i.e.,

$$\mathbf{h}_\tau^{t+1} = \mathbf{h}_\tau^t, \quad L_{t,\tau} = 0 \text{ and } t \neq \tau. \tag{27}$$

In this way, DAGG forms the representation $\mathbf{H}_{t+1} = (\mathbf{h}_1^{t+1}, \ldots, \mathbf{h}_t^{t+1})$ for the next step $t+1$.

An overview of the DAGG architecture is presented in Figure 4. This model offers several advantages. Compared to GraphRNN (You et al., 2018), DAGG uses attention instead of recurrent steps for computing the edge probabilities. This avoids the optimization issues of recurrent steps, such as vanishing gradients, making the model easier to optimize. Compared to other works (Li et al., 2018; Kawai et al., 2019), DAGG does not run a GNN at every step to compute the node representations of each partial graph. Instead, DAGG only updates the representations of nodes that are connected to the new node, so updating the node representations is more efficient.

6.3 Running time

Here we discuss the runtime complexity of the new methods introduced in this section. We first analyze the complexity of the new inference method, Rout-VI. The computation of node representations with a GNN takes time $\mathcal{O}(kmd)$, with k being the number of GNN layers, m being the number of edges, and d being the dimension of all the hidden vectors. Then, the sampling procedure takes time $\mathcal{O}(n^2)$, because each step needs to compare the context vector with all the nodes that have not been selected. Thus, to obtain S samples, the overall time is $\mathcal{O}(kmd + Sn^2)$, since the node representations are shared by all samples. Compared to the previous method ROS-VI (Chen et al., 2021), which takes time $\mathcal{O}(Snkmd + Sn^2)$, Rout-VI saves time by running the GNN only once and sharing it between samples, therefore, Rout-VI is significantly more efficient than ROS-VI.

We now discuss the complexity of the DAGG model. Generating a graph with n nodes and maximum degree δ takes time $\mathcal{O}(n^2 d + n\delta^2 d)$, where d is the dimension of all hidden vectors. This is because the computation of a new node's representation and that of sampling probabilities both take $\mathcal{O}(td)$ at each step t, then it takes time $\mathcal{O}(n^2 d)$ overall. The extra $n\delta^2 d$ term comes from the self-attention, which takes time $\mathcal{O}(\delta^2 d)$ at each step.

7. Experiments

In this section, we design a set of experiments to assess the performance of the new inference network Rout-VI and generative model DAGG. We also study the tightness of the variational lower bound. In particular, we apply Rout-VI over three existing generative models to analyze whether their performance improves. We also study the computation time of Rout-VI, which has lower time complexity than our previous inference network (Chen et al., 2021). We assess the performance of DAGG by combining it with Rout-VI. We also conduct extensive experiments to investigate the generation orders sampled from the variational distribution.

7.1 Experimental setup

Datasets. We use 8 datasets that are commonly used for benchmarking graph generative models: (1) *Community-small*: 1000 community graphs with $12 \leq |V| \leq 20$. Each graph has two communities generated from the model of Erdős and Rényi (1960). (2) *Citeseer-small*: 400 subgraphs with $5 \leq |V| \leq 20$, extracted from the Citeseer network (Sen et al., 2008) using random walk. (3) *Enzymes*: 600 protein graphs from the BRENDA database (Schomburg et al., 2004) with $10 \leq |V| \leq 125$. (4) *Lung*: 1000 chemical graphs with $6 \leq |V| \leq 50$, sampled from Kim et al. (2018). (5) *Yeast*: 1000 chemical graphs with $5 \leq |V| \leq 50$, sampled from Kim et al. (2018). (6) *Cora*: 1000 subgraphs with $9 \leq |V| \leq 97$, extracted from the Cora network (Sen et al., 2008) using random walk. (7) *SBM-assortative*: 1000 graphs generated by a stochastic block model (SBM) (Holland et al., 1983). We use three blocks, all with size 20. We generate the dataset by setting the probability of an edge between any two nodes to 0.3 if they are in the same block, or 0.05 otherwise. (8) *MMSBM*: 1000 graphs generated by a Mixed Membership Stochastic Blockmodel (MMSBM) (Godoy-Lorite et al., 2016). We set three communities with graph size 60. The block connectivity matrix, which defines the probability of interaction between nodes, is $[[0.4, 0.2, 0.3], [0.2, 0.1, 0.3], [0.3, 0.3, 0.4]]$; and the vector $[0.2, 0.1, 0.1]$ is the Dirichlet prior of nodes' membership probabilities.

Graphs in Lung and Yeast datasets reprent structures of chemical compounds. Graphs in the Enzymes dataset represent protein tertiary structures. While there is only one single graph in the Citeseer or Cora datasets, we sample subgraphs via random walk to form the corresponding datasets. We split all datasets into three parts: the train set (80%), validation set (10%), and test set (10%).

Methods. We compare DAGG against three recent graph generative models: GraphDF (Luo et al., 2021), GraphRNN (You et al., 2018), and GraphGEN (Goyal et al., 2020). We use their original training methods with default hyperparameters. GraphDF and GraphRNN use BFS orders for training.

To analyze the quality of Rout-VI, we also include two baselines to form the variational distribution: a uniform distribution over node orders, and our previous approach, ROS-VI (Chen et al., 2021). Both baselines are used to train GraphRNN and GraphGEN, and Rout-VI is used to train all four generative models. We compute the variational lower bound and estimate the log-likelihood for each setting. For ROS-VI and Rout-VI, we use the Nauty package (McKay and Piperno, 2013) to compute $|\Pi(A)|$ (see Section 4).

Table 1: Approximate test log-likelihood and ELBO of different graph generation models. Rout-VI improves the model fitting metrics, and DAGG fits the data better than competing methods when they are all trained with Rout-VI.

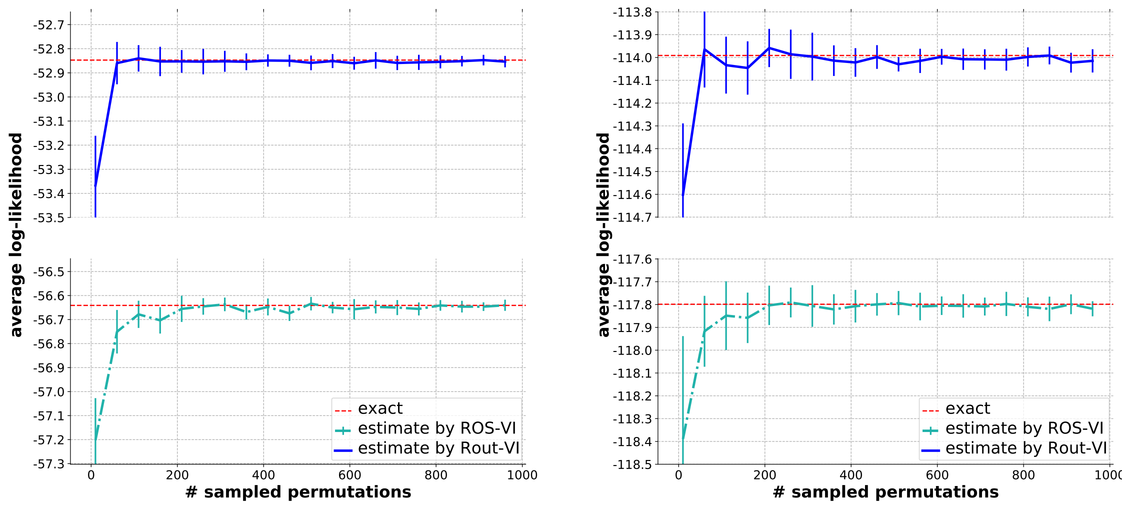
		Community-small log-like/ELBO	Citeseer-small log-like/ELBO	Enzymes log-like/ELBO	Lung log-like/ELBO
GraphDF	uniform	-120.3/-129.7	-181.4/-184.5	-410.2/-411.8	-122.7/-124.3
	Rout-VI	-103.1/-110.4	-93.5/-96.2	-250.7/-256.4	-120.4/-122.3
GraphRNN	uniform	-154.6/-157.6	-101.9/-105.7	-340.3/-349.1	-232.4/ -242.2
	ROS-VI	-53.7/-59.9	-89.6/-93.2	-274.9/-282.8	-155.9/-175.8
	Rout-VI	-25.2/-26.0	-39.2 / -46.6	-92.6/ -102.9	-78.3/-84.9
GraphGEN	DFS	-263.74/NA	-73.0/NA	-574.2/NA	-140.1/NA
	ROS-VI	-26.6/-35.0	-64.3/-71.1	-189.7/-213.8	-117.3/-125.5
	Rout-VI	-22.5/-31.8	-36.6/-40.2	-89.5/-93.1	-80.3/ -82.7
DAGG	Rout-VI	**-21.5/-22.8**	**-27.4/-31.6**	**-75.1/-82.6**	**-62.3/-63.6**
		Yeast log-like/ELBO	Cora log-like/ELBO	SBM-assortative log-like/ELBO	MMSBM log-like/ELBO
GraphDF	uniform	-140.0/-141.7	-243.4/-246.9	-183.5/-191.8	-237.8/-245.6
	Rout-VI	-72.5/-73.1	-146.7/-148.6	-137.4/-145.7	-214.5/-219.1
GraphRNN	uniform	-189.3/-200.1	-380.6/-401.8	-162.4/-171.1	-192.3/-203.5
	ROS-VI	-109.1/-133.7	-345.3/-358.3	-90.4/-95.7	-154.0/-158.8
	Rout-VI	-55.2/-58.9	-142.3/-154.8	-78.6/-82.4	-149.5.2/- 155.7
GraphGEN	DFS	-66.4/NA	-199.5/NA	-210.4/NA	-188.4/NA
	ROS-VI	-64.9/-72.3	-143.6/-152.3	-87.2/-91.1	-147.1/-154.2
	Rout-VI	-55.4/-61.2	-111.4/-122.8	-84.0/-88.3	-125.5./-138.9
DAGG	Rout-VI	**-50.4/-52.4**	**-108.5/-117.4**	**-72.2/-76.1**	**-110.7/-117.4**

7.2 Evaluation of model fitting

We evaluate different models by their predictive log-likelihood. For each model, we use $L = 1{,}000$ samples to estimate the test log-likelihood via importance sampling (Eq. 13), using the variational distribution $q_\phi(\pi \mid G)$ as the proposal. We also report the variational lower bound (ELBO) from Eq. 9 to assess the tightness of the bound. We estimate both quantities 10 times and compare methods via t-test at the 5% significance level. The results are in Table 1. The standard deviations are very small (less than 0.01) and not included here. Note that the original GraphGEN is trained with an approximate canonical order $\pi^\star$ derived from DFS (as discussed in Section 4, the likelihood of GraphGEN is problematic because it differs from the probability of sampled graphs).

We see that generative models trained with the two inference networks exhibit better predictive performance than those with a uniform variational distribution. The improvements are often significant. Compared to ROS-VI, Rout-VI further improves model fitting on all datasets. We can also see that the ELBO is relatively tight for most cases. These results show evidence that VI is an effective method for training graph generative models, and Rout-VI is a strong way to construct the inference network.

We also observe that DAGG outperforms the baseline generative models when they are all trained with Rout-VI. Therefore, among all the methods being compared, the DAGG model combined with the Rout-VI training is the one that performs the best.



Yeast

Lung

Figure 5: Comparison of the estimated log-likelihood and the exact log-likelihood for 5 small graphs from the Yeast dataset (left) or from the Lung dataset (right). An estimation with $L = 400$ importance samples is close to the exact log-likelihood.

To understand the performance improvement of Rout-VI over ROS-VI, we examine the node orders inferred from both methods. On the Yeast dataset, the GraphGEN trained with a DFS order has very good performance, close to the models trained with VI. We check the node orders sampled from ROS-VI and Rout-VI and discover that they are very similar to DFS orders; this indicates that VI can automatically discover that DFS is a good order for this dataset. In contrast, on the Community-small dataset, VI improves significantly over the DFS order of GraphGEN. On this dataset, DFS orders are not good choices, and the learned variational distribution is able to avoid such node orders.

Finally, we study the accuracy of the importance sampling evaluation of the model's log-likelihood through a separate experiment. In particular, we compute the exact log-likelihood of a few small graphs by enumerating all possible permutations (the computation is feasible for small graphs). We randomly choose 5 graphs with 6 nodes from Yeast and 5 graphs with 8 nodes from Lung. To avoid any possible bias brought by different generative models, we only consider one model, GraphRNN, trained with the two VI methods (ROS-VI and Rout-VI). Figure 5 shows the results on the two datasets. When the number of samples L approaches 1,000, the gap between the exact log-likelihood and the estimated log-likelihood becomes very small. We conclude that the estimation through importance sampling can be reliably used for model selection and comparison.

Table 2: The MMD evaluation of graphs generated by different methods. Among different methods, the DAGG model combined with Rout-VI in general generate graphs that are most similar to the original graphs.

		GraphDF		GraphRNN				GraphGEN			DAGG
		uniform	Rout-VI	BFS	uniform	ROS-VI	Rout-VI	DFS	ROS-VI	Rout-VI	Rout-VI
Community -small	Deg.	0.063	0.055	0.034	0.096	**0.018**	0.021	0.070	0.143	0.094	0.019
	Clust.	**0.012**	0.093	0.114	0.091	0.014	0.014	0.931	0.248	0.152	0.013
	Orbit	0.037	0.024	0.009	0.021	0.008	**0.007**	0.178	0.068	0.047	0.008
	Short.	0.059	0.052	0.014	0.018	0.007	**0.005**	0.142	0.065	0.069	0.011
Citeseer -small	Deg.	0.082	0.065	0.016	**0.009**	0.080	0.026	0.047	0.032	0.020	0.012
	Clust.	0.147	0.113	0.050	0.090	0.050	0.033	0.032	0.078	0.041	**0.028**
	Orbit	0.018	0.011	0.004	0.003	**0.002**	0.003	0.017	0.008	0.008	**0.002**
	Short.	0.026	0.022	0.050	0.005	0.008	**0.004**	0.029	0.012	0.007	**0.004**
Enzyme	Deg.	0.056	0.040	0.034	0.042	0.015	0.012	0.716	0.346	0.199	**0.011**
	Clust.	0.187	0.155	0.085	0.104	0.067	0.054	0.456	0.440	0.268	**0.052**
	Orbit	0.049	0.023	0.043	0.074	0.023	**0.017**	0.078	0.020	0.018	0.018
	Short.	0.032	0.026	0.611	0.048	0.047	0.025	0.055	0.048	0.023	**0.021**
Lung	Deg.	0.043	0.038	0.103	1.213	0.074	0.059	0.049	0.022	**0.015**	**0.015**
	Clust.	0.064	0.055	0.301	**0.002**	0.060	0.041	0.017	0.008	0.007	0.008
	Orbit	0.033	0.029	0.043	0.081	0.004	0.004	**0.000**	**0.000**	**0.000**	0.001
	Short.	0.028	0.041	0.054	0.088	0.004	**0.002**	**0.002**	0.003	**0.002**	**0.002**
Yeast	Deg.	0.440	0.128	0.512	0.746	0.097	0.024	0.014	0.012	0.010	**0.009**
	Clust.	0.285	0.084	0.153	0.351	0.092	0.031	**0.003**	**0.003**	**0.003**	**0.003**
	Orbit	0.045	0.039	0.026	0.070	0.005	0.003	**0.000**	**0.000**	0.001	0.001
	Short.	0.077	0.035	0.016	0.052	0.007	0.004	0.004	0.008	0.005	**0.003**
Cora	Deg.	0.556	0.100	1.125	0.188	0.066	0.049	0.099	0.056	0.049	**0.039**
	Clust.	0.381	0.256	1.002	0.206	0.171	0.100	0.167	0.103	0.100	**0.087**
	Orbit	0.113	0.072	0.427	0.200	0.052	0.062	0.122	0.069	0.062	**0.041**
	Short.	0.090	0.087	0.518	0.231	0.045	0.032	0.147	0.086	0.043	**0.025**
SBM-assort.	Deg.	0.076	0.053	0.064	0.128	0.042	0.037	0.085	0.079	0.037	**0.035**
	Clust.	0.024	**0.022**	0.056	0.142	0.033	0.031	0.080	0.116	0.038	0.028
	Orbit	0.051	0.046	0.023	0.042	0.017	0.017	0.062	0.033	0.029	**0.014**
	Short.	0.034	0.030	0.012	0.025	**0.006**	0.009	0.085	0.042	0.020	0.009
MMSBM	Deg.	0.113	0.097	0.058	0.151	0.056	**0.046**	0.950	0.061	0.054	0.052
	Clust.	0.126	0.122	**0.039**	0.174	0.041	0.045	0.113	0.047	0.045	0.040
	Orbit	0.246	0.196	0.045	0.089	0.037	0.034	0.061	0.049	0.042	**0.031**
	Short.	0.055	0.055	0.031	0.073	**0.025**	**0.025**	0.038	0.041	0.029	0.026

7.3 Evaluation of graph generation

Here we 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 in terms of the following network properties: the degree distribution (Deg.), clustering coefficients (Clust.), occurrences of 4-node orbits (Orbit), and pairwise shortest distances (Short.). For each graph property, we extract a histogram (e.g., of node degrees) from a test graph or a generated graph. Then we compare two groups of histograms respectively from test graphs and generated graphs using MMD (Gretton et al., 2012) (lower MMD is better). We choose to use MMD metrics because we can have a direct comparison with previous neural methods. Our evaluation covers several important network properties (Newman, 2003): the MMD metrics computed from node degrees, clustering coefficients, and shortest distances can respectively evaluate the scale-free effect, the transitivity, and the small-world effect of generated graphs.

Table 2 shows the MMD evaluation on the eight datasets. We indicate the best performance(s) across all methods with bold numbers. For each generative model, we indicate the best inference method by underlining the performance number. We first check the performance improvement from the new inference method. If three MMD metrics are improved

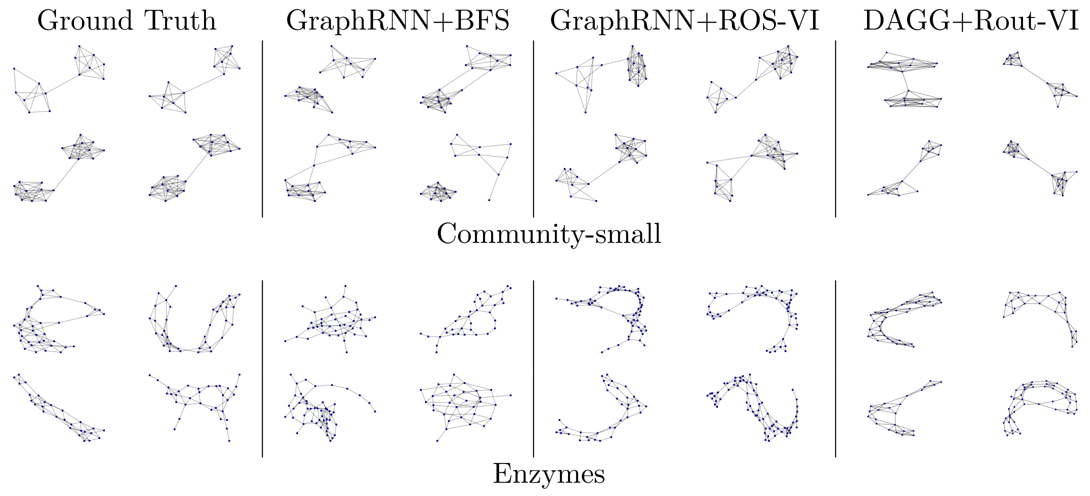


Figure 6: Graph samples from different models trained on Community-small and Enzymes. The DAGG model fitted with Rout-VI generates graphs that have structural patterns similar to the real data.

on a dataset, we consider the new method has improvement on that dataset. We see that Rout-VI improves the performance of GraphRNN on all datasets except Citeseer-small. For GraphDF, the learned order distribution is better than uniform distribution on all datasets. For GraphGEN, Rout-VI improves the performance on all datasets except Community-small and Yeast. On the Yeast dataset, the results for GraphGEN are similar regardless of the choice of node orders; this is consistent with the previous results on the log-likelihood. Regarding the generative model, DAGG combined with Rout-VI performs the best on four datasets (Citeseer-small, Enzyme, Yeast, and Cora) across all combinations. Its performance on the other two datasets is only slightly worse than the best performance across all other approaches. Overall, the results indicate that autoregressive generative models trained with VI produce graphs of higher quality than those trained with ad-hoc node orders. The proposed DAGG combined with Rout-VI is a strong generative model.

We also compare the generated graphs of different methods. For this, we focus on GraphRNN and DAGG. Figure 6 (left) shows four graphs from the Community-small dataset and four graphs from the Enzymes dataset. Figure 6 then shows graphs generated by variants of GraphRNN that are trained with BFS orders and ROS-VI. Figure 6 (right) also shows the graphs generated by DAGG with Rout-VI. For Community-small, the two VI methods capture the graph patterns—only one edge exists between the two communities, with the samples from DAGG being closer to the ground truth. In contrast, GraphRNN+BFS generates some graphs with multiple or no edges between the two communities. For the Enzymes dataset, the samples from the two VI methods are also more similar to the ground truth data than the BFS orders—they have the shape of long strips, and some of them contain large cycles. Both VI methods generate graphs of similar quality (this is consistent with the MMD metrics for cluster and orbit on the Enzymes dataset for both methods being close).

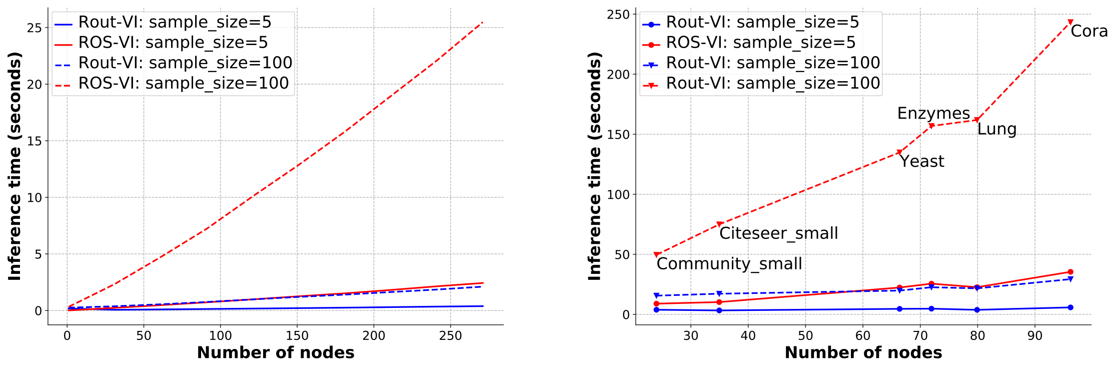


Figure 7: Comparison of the inference time of the two VI methods. **(Left)** Inference time on cycle graphs. **(Right)** Inference time on 100 graphs from the 6 datasets. Rout-VI is significantly faster than ROS-VI.

7.4 Inference time

For a graph $G = (V, E)$, we have discussed the computational complexity of ROS-VI and Rout-VI in Section 6.3. Here we empirically assess their inference time. Note that the time of sampling node orders is roughly the same as that of computing probabilities, so we consider sampling time here.

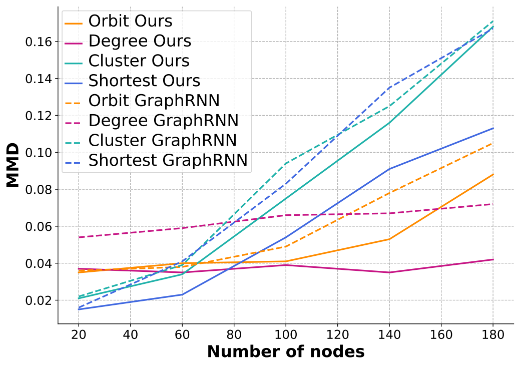
We test the running time on controlled graphs (cycle graphs) and real datasets. We use graphs of different sizes: for cycle graphs, we vary the number of nodes; for real datasets, we randomly take 100 graphs from each dataset and report their average size. We apply both inference networks to sample 5 or 100 node orders for each graph. We repeat the process 10 times and report the average running time. Both methods run on an RTX 3080 GPU.

Figure 7 shows the running time of the two inference methods under different settings. We see that Rout-VI significantly accelerates the inference procedure on cycle graphs and graphs from real datasets. We find that the time it takes for ROS-VI to obtain 5 samples is about the same as it takes for Rout-VI to obtain 100 samples; that is, Rout-VI exhibits approximately 20-times speed-up over Rout-VI. As discussed in Section 6.3, this is because Rout-VI requires fewer GNN runs.

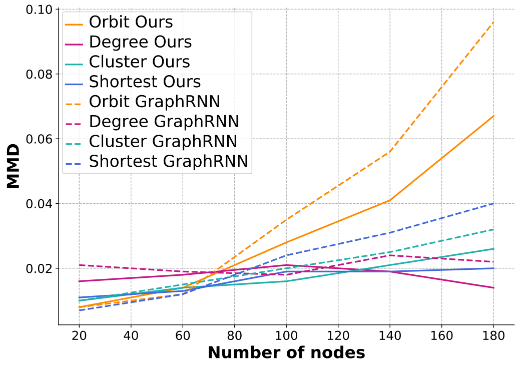
7.5 Generation quality and graph sizes

It becomes increasingly challenging for generative autoregressive models to capture graph structures when the graph size becomes large. Here we investigate the relationship between the performance of our model and graph sizes.

We construct sets of graphs of different sizes to evaluate generative models. We consider two datasets here. We first sample subgraphs with different graph sizes from the Cora network (Sen et al., 2008). We then sample two-community graphs of different sizes from the Community-small dataset. For each graph size, we train and test our autoregressive model and use GraphRNN as a baseline model.



Cora



Community

Figure 8: MMD performance as a function of the graph size.

We plot our model's performances against graph sizes in Figure 8. Both our model and GraphRNN are able to maintain the degree distributions of real graphs, as evidenced by the MMD-degree metrics being relatively constant across graph sizes. However, generated graphs have difficulties to capture the graph orbits and clustering coefficients of the graphs from the test set: the MMD-orbit and MMD-cluster metrics become worse as the graph size increases.

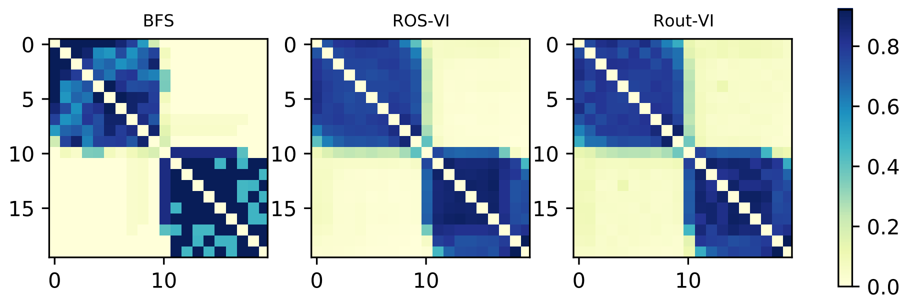
We hypothesize that a node's degree is a local property that is only related to incident edges and thus is easier to learn. However, MMD-orbit and MMD-cluster require the model to learn higher order properties such as triangles and squares, and these properties are harder to learn as the graph size increases. These results show some evidence that our autoregressive generative model has difficulties in capturing non-local graph properties when graphs are large.

We also find that autoregressive models become slow when modeling large graphs, which is already indicated in the complexity analysis: the model need at least $\mathcal{O}(n^2)$ time to generate a graph because it needs to generate all entries in the adjacency matrix.

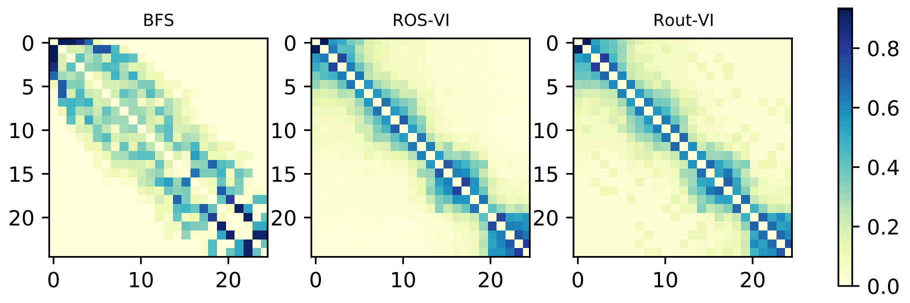
7.6 Analysis of the inferred node orders

In this section, we study the order sampled from the variational distribution $q_\phi(\pi \mid G)$ to gain insights on how the VI approach improves training.

We first visualize the adjacency matrices corresponding to different node orders in Figure 9. We choose one graph from the Community-small dataset and one graph from the Enzymes dataset. For each graph, we sample node orders from three distributions (BFS ordering, ROS-VI, and Rout-VI) and plot the average of the adjacency matrices corresponding to these node orders. (We choose GraphRNN as the generative model, which was fitted using each of the three approaches for obtaining node orders.) The average of these matrices indicates the marginal distribution of each entry of the adjacency matrix. On Community-small (top), the BFS order seems to be good for model training: adjacency matrices from these orders have near-zero anti-diagonal blocks, which are easy for the



Community-small



Enzymes

Figure 9: Average of sampled adjacency matrices corresponding to different node orders.

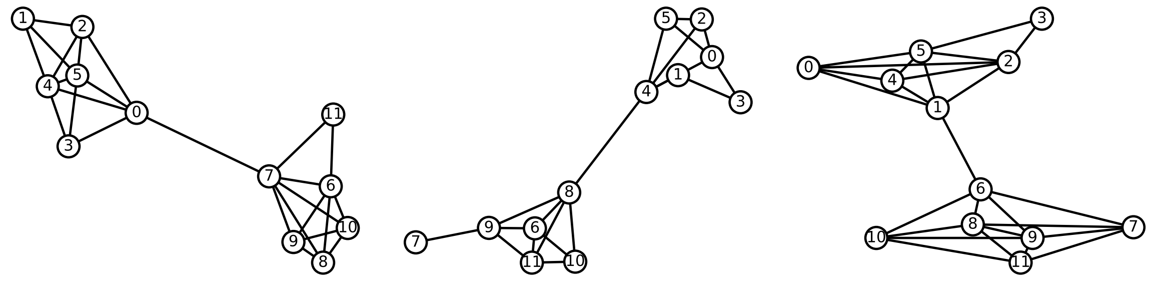
generative model to fit. The variational distributions learned by ROS-VI and Rout-VI discover this pattern. On the Enzymes dataset (bottom), the two VI methods learn to form band matrices. In contrast, BFS scatters non-zeros to a wider range, which likely results in patterns that are harder to learn for the generative model.

Comparing ROS-VI and Rout-VI, Rout-VI tends to have more non-zero off-diagonal entries. In fact, we have observed that Rout-VI tends to first generate the main part of a graph and then do some decorations. We do not have a thorough understanding why the generative models benefit from such property, and we leave it as an open question.

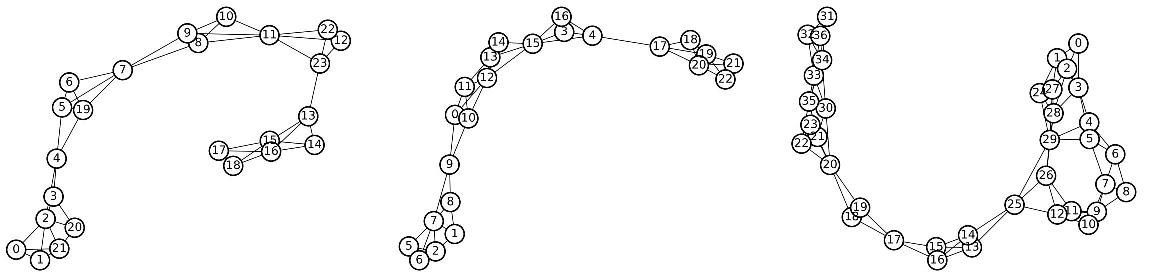
Finally, we visualize the orders learned by Rout-VI in Figure 10 on some graphs from the dataset. To make this plot, we fist obtain $q_\phi(\pi \mid G)$ and then greedily choose each node with the largest probability in the sequence. We plot the node orders for three graphs from the Community-small and three graphs from the Enzymes dataset. We observe that the generation of a graph from Community-small always strictly finishes one community before moving on to the next (like BFS). The generation of a graph from the Enzymes dataset tends to form the backbone first (like DFS) and then decorate the backbone with more nodes. This is consistent with the results in Figure 9 discussed above.

8. Conclusions and Discussion

In this paper, we have analyzed the likelihood of autoregressive graph generative models. A naive evaluation of the likelihood would require to sum over all possible adjacency matrices



Community-small



Enzymes

Figure 10: Orders greedily obtained from the approximate posterior of Rout-VI on graphs from Community-small and Enzymes dataset.

of the graph. Instead, we provide an approach that sums over node orders—a summation variable much easier to enumerate. In doing so, we have also revealed the relation between probability calculations and graph automorphisms.

To avoid the intractable summation in the likelihood, we have used VI. In particular, we have formed a lower bound on the log-likelihood by building a variational distribution over the node orders. The lower bound can be estimated with a small number of samples, allowing for efficient model training. Moreover, the VI method works for any graph generative model that specifies a distribution of adjacency matrices. We can understand the VI approach from another angle: the variational distribution is optimized to provide good adjacency matrices to the generative model; therefore, the model can be trained better than if it only had access to a single or small number of pre-determined adjacency matrices.

We have designed a new inference network, Rout-VI, which makes use of routing search to greatly speed up the inference over previous approaches. We have also introduced a graph generative model, DAGG, that uses attention mechanisms. Our experimental results show that Rout-VI and DAGG improve the performance in both data fitting and graph generation. When using a fixed inference method, DAGG outperforms other generative models.

Limitations. Despite the empirical advantages demonstrated in Section 7, DAGG also exhibits some limitations when modeling large graphs. First, it is only efficient for small and moderately-sized graphs; this limitation is shared with other autoregressive models. Second, as the graph size increases, DAGG struggles to capture the high-level properties of the graph (see Section 7.5); this limitation is also shared with existing models such as GraphRNN.

Future work. There are some potential avenues for future work. First, we can extend our inference component so that it can work with generative models that add a batch of nodes (Liao et al., 2019) or a substructure (Jin et al., 2020) to the graph at each step. We expect that inferring the node orders for these models will improve data fitting in the same way as this work has shown. Second, we can extend our generative model to add multiple nodes or subgraph structures at each generative step. Based on the attention mechanism, we can still update representations of nodes or substructures to model sequential decisions of expanding a graph. This extension may address the two main limitations discussed above.

Acknowledgements

We thank Yujia Li for his insightful comments. We also thank the editor and anonymous reviewers for their constructive feedback. Li-Ping Liu has been supported by NSF grants 1908617 and 2239869. Xu Han and Xiaohui Chen have been supported by Tufts RA support.

References

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

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

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

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

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

X. Chen, X. Han, J. Hu, F. Ruiz, and L. Liu. Order matters: Probabilistic modeling of node sequence for graph generation. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 1630–1639. PMLR, 18–24 Jul 2021. URL `http://proceedings.mlr.press/v139/chen21j.html`.

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

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

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

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

A. Frieze and M. Karoński. *Introduction to Random Graphs*. Cambridge University Press, 2015. doi: 10.1017/CBO9781316339831.

A. Godoy-Lorite, R. Guimerà, C. Moore, and M. Sales-Pardo. Accurate and scalable social recommendation using mixed-membership stochastic block models. *Proceedings of the National Academy of Sciences*, 113(50):14207–14212, 2016.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

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

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

T. L. Griffiths and Z. Ghahramani. The Indian buffet process: An introduction and review. *The Journal of Machine Learning Research*, 12:1185–1224, 2011.

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

F. Harary, E. Palmer, and M. U. A. A. D. of MATHEMATICS. *Graphical Enumeration*. Academic Press, 1973. ISBN 9780123242457. URL `https://books.google.com/books?id=yqr98zXOalcC`.

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

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

W. Jin, R. Barzilay, and T. Jaakkola. Hierarchical generation of molecular graphs using structural motifs. In *International conference on machine learning*, pages 4839–4848. PMLR, 2020.

C. K. Joshi, T. Laurent, and X. Bresson. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*, 2019.

C. K. Joshi, Q. Cappart, L.-M. Rousseau, T. Laurent, and X. Bresson. Learning tsp requires rethinking generalization. *arXiv preprint arXiv:2006.07054*, 2020.

W. Kawai, Y. Mukuta, and T. Harada. Scalable generative models for graphs with graph attention mechanism. *arXiv preprint arXiv:1906.01861*, 2019.

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

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

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

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

W. Kool, H. van Hoof, and M. Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2018.

W. Kool, H. van Hoof, and M. Welling. Buy 4 reinforce samples, get a baseline for free! 2019.

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

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

P. Lippe and E. Gavves. Categorical normalizing flows via continuous transformations. In *International Conference on Learning Representations*, 2020.

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

T.-Y. Liu et al. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331, 2009.

Y. Luo, K. Yan, and S. Ji. Graphdf: A discrete flow model for molecular graph generation. In *International Conference on Machine Learning*, pages 7192–7203. PMLR, 2021.

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

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

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

G. Mena, D. Belanger, S. Linderman, and J. Snoek. Learning latent permutations with gumbel-sinkhorn networks. *arXiv preprint arXiv:1802.08665*, 2018.

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

M. E. Newman. The structure and function of complex networks. *SIAM review*, 45(2): 167–256, 2003.

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

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

P. Orbanz and D. M. Roy. Bayesian models of graphs, arrays and other exchangeable random structures. *IEEE transactions on pattern analysis and machine intelligence*, 37 (2):437–461, 2014.

A. B. Owen. *Monte Carlo theory, methods and examples.* 2013.

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

M. Paulus, D. Choi, D. Tarlow, A. Krause, and C. J. Maddison. Gradient estimation with stochastic softmax tricks. *Advances in Neural Information Processing Systems*, 33: 5691–5704, 2020.

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

L. Richter, A. Boustati, N. Nüsken, F. Ruiz, and O. D. Akyildiz. Vargrad: a low-variance gradient estimator for variational inference. *Advances in Neural Information Processing Systems*, 33:13481–13492, 2020.

T. Salimans and D. A. Knowles. On using control variates with stochastic approximation for variational Bayes and its connection to stochastic linear regression. *arXiv preprint arXiv:1401.1022*, 2014.

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

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

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

M. Simonovsky and N. Komodakis. GraphVAE: Towards generation of small graphs using variational autoencoders. In *International Conference on Artificial Neural Networks*, 2018.

T. A. Snijders, P. E. Pattison, G. L. Robins, and M. S. Handcock. New specifications for exponential random graph models. *Sociological methodology*, 36(1):99–153, 2006.

Y. W. Teh, D. Görür, and Z. Ghahramani. Stick-breaking construction for the Indian buffet process. In *International Conference on Artificial Intelligence and Statistics*, 2007.

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

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. 2017b.

H. Wang, J. Wang, Jia Wang, M. Zhao, W. Zhang, F. Zhang, X. Xie, and M. Guo. GraphGAN: Graph representation learning with generative adversarial nets. In *AAAI Conference on Artificial Intelligence*, 2018.

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

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

Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

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

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