A Simple Latent Variable Model for Graph Learning and Inference

Manfred Jaeger
Aalborg University
jaeger@cs.aau.dk

Antonio Longa
University of Trento
antonio.longa@unitn.it

Steve Azzolin
University of Trento
steve.azzolin@unitn.it

Oliver Schulte
Simon Fraser University
oschulte@cs.sfu.ca

Andrea Passerini
University of Trento
andrea.passerini@unitn.it

Abstract

We introduce a probabilistic latent variable model for graphs that generalizes both the established graphon and stochastic block models. This *naive histogram AHK model* is simple and versatile, and we demonstrate its use for disparate tasks including complex predictive inference usually not supported by other approaches, and graph generation. We analyze the tradeoffs entailed by the simplicity of the model, which imposes certain limitations on expressivity on the one hand, but on the other hand leads to robust generalization capabilities to graph sizes different from what was seen in the training data.

1 Introduction

Most state-of-the-art machine learning techniques for graph data are highly specialized towards specific tasks such as node classification or synthetic graph generation. This is the case, in particular, for the graph neural network technology. In contrast, one may be interested in more flexible approaches that support a variety of different inference tasks based on a single model. This is the tradition of probabilistic graphical models [16], which have been adapted to the handling of graph data in the field of *statistical relational learning (SRL)* by frameworks like relational Bayesian networks [12], Markov logic networks [21], or ProbLog [7]. Conceptually extremely powerful, these SRL approaches face numerous challenges in the form of computational complexity of inference and learning, and the need to provide some structural model specifications based on domain expertise. One cause for the high computational complexity of SRL models lies in the fact that the answer to a probabilistic query involving a certain set of named entities often depends on how many other entities there exist in the domain, even if nothing is known about their properties, and their connections to the query entities. In the worst case, the complexity of computing a query is exponential in the number of these additional entities [11]. Technically, these SRL models are not *projective* in the sense of Shalizi and Rinaldo [23].

Investigating projectivity in a more general setting of multi-relational graphs, Jaeger and Schulte [13] have introduced the *AHK* model, which can be seen as a generalization of the classic *graphon* model for random graphs developed by Aldous [3], Hoover [10], and Kallenberg [14]. As introduced in [13], the AHK model is a very general, abstract mathematical model that does not directly provide algorithmic learning and inference solutions. In this paper we develop a concrete, limited, class of AHK models, together with practical learning and inference approaches. We call this model class the *naive histogram AHK (NH-AHK$^-$)* model, because it incorporates a probabilistic independence assumption similar in nature to the naive Bayes assumption, and it employs non-parametric histogram-based specifications of conditional probability distributions.

Our model class can be seen as a lightweight alternative to SRL frameworks. Compared to the latter, it loses a substantial amount of modeling capabilities (though, as we shall see, it also provides

Jaeger et al., A Simple Latent Variable Model for Graph Learning and Inference. *Proceedings of the Second Learning on Graphs Conference (LoG 2023)*, PMLR 231, Virtual Event, November 27–30, 2023.

some capabilities not found in current SRL), but gains in terms of computational tractability, and does not depend on model components defined using elements of logic or programming languages, which are very difficult to learn fully automatically from data. Compared to many other machine learning techniques, the naive histogram AHK model retains the capability to answer a wide range of probabilistic queries, to handle incomplete data both in learning and inference, and to function as a generator for synthetic graphs.

We introduce the NH-AHK$^-$ model in Section 4 and discuss some basic theoretical properties in Section 5. The main algorithmic challenges and solutions are described in Section 6. In Section 7 we demonstrate the range of possible applications of the NH-AHK$^-$ model for graph modeling and reasoning, and graph generation. An implementation of NH-AHK$^-$ and scripts for the reproduction of our experiments are available at `https://github.com/manfred-jaeger-aalborg/AHK`.

2 Related Work

Several authors have proposed methods for learning a graphon model [2, 4, 9, 24]. Often the graphon is required to be given by a smooth function [4, 9, 18, 24], but histograms or stochastic block models are also often considered, at least as an approximation [2, 9]. Learning is almost exclusively performed for descriptive analysis of a single network, and the learned graphon is often presented as a heatmap representation of the network [4, 24]. Applications to predictive inference then are limited to transductive link prediction [18]. Graphon learning from multiple graphs has been considered under the assumption that training data contains graphs over the same set of nodes, which all have a fixed setting of their latent variables in repeated realizations of the random graph structure [2]. Earlier work on graphon learning only considered graphs without node attributes. Numeric node attributes have recently been considered in [5, 25]. All of these works differ substantially from ours in that they do not consider graphon learning from multiple graphs for applications in inductive predictive inference and graph generation.

3 Basic Definitions

Tuples. The set of integers $\{0, \ldots, n-1\}$ is denoted $[n]$. For any $d \geq 1$, $[n]^d$ then is the set of d-tuples with elements from $[n]$. With $[n]^d_{\neq}$ we denote the set of d-tuples with all distinct elements. We write $\langle n \rangle^d$ for the set of d-tuples in which elements appear in their natural order, and $\langle n \rangle^d_{\neq}$ for ordered tuples with all distinct elements. Thus, $\langle n \rangle^d$ and $\langle n \rangle^d_{\neq}$ are the outcome spaces of drawing an unordered sample of size d from $[n]$, with and without replacement, respectively. Tuples are denoted in boldface letters $\boldsymbol{u}, \boldsymbol{b}, \ldots$, and components of such tuples by corresponding subscripted normal face letters $u_p, b_i, \ldots$.

Graphs. We only consider graphs without numeric attributes or edge weights. A *signature* $S = \mathcal{A} \cup \mathcal{E}$ then consists of categorical node attributes $\mathcal{A} = \{A_0, \ldots, A_{|\mathcal{A}|-1}\}$ and edge relations $\mathcal{E} = \{E_0, \ldots, E_{|\mathcal{E}|-1}\}$. A *S-graph* $\omega = ([n], \boldsymbol{A}, \boldsymbol{E})$ consists of a set of nodes $[n]$ for some $n \in \mathbb{N}$, an $n \times |\mathcal{A}|$-dimensional matrix $\boldsymbol{A}$ containing the attribute values of all nodes, and a $n \times n \times |\mathcal{E}|$-dimensional tensor $\boldsymbol{E}$ where $\boldsymbol{E}[:,:,l]$ is the adjacency matrix for relation E_l. We refer to n as the *size* of ω. We denote by $\Omega^{(n)}$ the set of all graphs for a given signature S with domain $[n]$. The relevant signature is usually implicit from the context, and not made explicit in the notation. An *incomplete graph* ω is a graph with ? entries in $\boldsymbol{A}$ or $\boldsymbol{E}$. We denote by $\Delta\Omega^{(n)}$ the set of probability distributions over $\Omega^{(n)}$. A *random graph model* defines for every $n \in \mathbb{N}$ a distribution $P_n \in \Omega^{(n)}$.

Permutations. $\mathcal{P}[n]$ stands for the set of permutations of $[n]$. For $n = 2$, this set contains two elements: the identity mapping, and the permutation with $\pi(0) = 1$. For the latter permutation we use the special symbol $\pi_{0 \leftrightarrow 1}$.

Types. We introduce the key concept of types. While the exact definitions are a bit technical, the underlying intuition is very simple: the 1-type of a node represents all the data available for that node in isolation, which consists of its attribute values, and its possible self-loops of relations. The (strict) 2-type of a pair of nodes contains all the data about edges connecting the two nodes. Both 1- and 2-types can again be represented as graphs over the canonical domains [1] and [2], respectively.

Formally: for a graph ω of size n, and $i \in [n]$, we denote with $\omega \downarrow i$ the graph $([1], \boldsymbol{A}[i,:], \boldsymbol{E}[i,i,:])$. We also call $\omega \downarrow i$ the *1-type* of i in ω, and denote by $\mathcal{T}_1$ the space of all possible 1-types. For a pair $(i,j) \in [n]^2_{\neq}$ we define $\omega \downarrow (i,j)$ as the graph $([2], \emptyset, \boldsymbol{E}_{i,j})$, where $\boldsymbol{E}_{i,j}$ stands for the $2 \times 2 \times |\mathcal{E}|$-dimensional tensor in which for $k < |\mathcal{E}|$: $\boldsymbol{E}_{i,j}[0,1,k] = \boldsymbol{E}[i,j,k]$, $\boldsymbol{E}_{i,j}[1,0,k] = \boldsymbol{E}[j,i,k]$, and $\boldsymbol{E}_{i,j}[0,0,k] = \boldsymbol{E}_{i,j}[1,1,k] = 0$. We also call $\omega \downarrow (i,j)$ the *strict 2-type* of (i,j) in ω, and denote by $\mathcal{T}_2$ the space of all strict 2-types (we add the qualifier "strict" in order to emphasize that these 2-types omit the information about i, j that is already contained in their respective 1-types). There is a one-to-one correspondence between graphs and mappings of nodes i and pairs (i,j) to 1-types and strict 2-types, respectively. The class of probabilistic models introduced in the following makes essential use of this correspondence.

Projectivity. A random graph model is projective, if for all $n_0 < n_1 \in \mathbb{N}$ the following holds: P_{n_0} is equal to the marginal distribution of P_{n_1} on induced subgraphs of size n_0 (see [13] for a full technical definition). Projectivity has important benefits: for inference, a query related to a specific set of nodes does not depend on the size of the graph the nodes are embedded in (assuming no further information on the nodes other than those named in the query is given). For learning: if the actual data-generating distribution can be represented by a member of a certain class of projective random graph models (such as the class of NH-AHK$^-$ models we introduce below), then the model learned from a full dataset (consisting of possibly very large graphs), and the model learned from a dataset consisting of (smaller) sampled induced sub-graphs of size n will agree on the probabilities for graphs up to size n (in the large sample limit, and assuming the true maximum likelihood solution is identified).

4 The Naive Histogram AHK$^-$ Model

We formally introduce our model. We start by recapitulating the general definition of [13] constrained to signatures and graphs as introduced in Section 3, and limited to what in [13] is called the AHK$^-$ model, which lacks an additional global latent mixture variable of the full AHK model. In the following $\Delta\mathcal{T}_i$ stands for the set of probability distributions over $\mathcal{T}_i$ ($i = 1, 2$).

Definition 4.1 Let S be a signature. An AHK$^-$-model for S and graphs of size n is given by

- A family of i.i.d. random variables $\{U_i | i \in [n]\}$, where each U_i is uniformly distributed on $[0,1]$.
- For $i \in [n]$: random variables T_i with values in $\mathcal{T}_1$, and for $(i,j) \in \langle n \rangle^2$: random variables $T_{(i,j)}$ with values in $\mathcal{T}_2$.
- A measurable function
$$f_1 : [0,1] \to \Delta\mathcal{T}_1, \tag{1}$$
and a measureable function
$$f_2 : [0,1]^2 \to \Delta\mathcal{T}_2 \tag{2}$$
that is permutation equivariant in the following sense: for $(u,u') \in [0,1]^2$ and $t \in \mathcal{T}_2$: $f_2(u,u')(t) = f_2(u',u)(\pi_{0\leftrightarrow 1}t)$.

The AHK$^-$-model defines a probability distribution over graphs $\Omega^{(n)}$ with n nodes as the expectation over latent u_i variables

$$P(\omega) = \int_{\boldsymbol{u} \in [0,1]^n} \prod_{i \in [n]} f_1(u_i)(\omega \downarrow i) \prod_{(i,j) \in \langle n \rangle^2} f_2(u_i, u_j)(\omega \downarrow (i,j)) d\boldsymbol{u}. \tag{3}$$

A simple strategy for satisfying the permutation equivariance condition for f_2 is to only define a function $f_2^<(u,u')$ for arguments $u < u'$, and let

$$f_2(u,u') = \begin{cases} f_2^<(u,u') & \text{if } u < u' \\ \pi_{0\leftrightarrow 1} f_2^<(u',u) & \text{if } u' < u. \end{cases} \tag{4}$$

With f_2 given in this form, and denoting by $\langle [0,1] \rangle^n$ the set of all ordered n-tuples of values from $[0,1]$, we can re-write (3) by replacing the integral over $[0,1]^n$ with a combination of a sum over

node permutations, and an integral only over $\langle[0,1]\rangle^n$.

$$P(\omega) = \sum_{\pi \in \mathcal{P}[n]} \int_{\boldsymbol{u} \in \langle[0,1]\rangle^n} \prod_{p \in [n]} f_1(u_p)(\omega \downarrow \pi(p)) \prod_{(p,q) \in \langle n \rangle^2} f_2^<(u_p, u_q)(\omega \downarrow (\pi(p), \pi(q)))d\boldsymbol{u}. \quad (5)$$

We use p, q to denote positions in a permutation π of $[n]$. Then $\pi(p)$ stands for the node $i \in [n]$ with the pth smallest U_i.

In order to turn this purely mathematical construction into an operational model, we need to design suitable classes of functions f_1 and $f_2^<$ for which learning and inference methods can be developed. In the following, we limit our descriptions and definitions to the $f_2^<$ functions. The f_1 are treated in an analogous manner. We base our design on two assumptions:

A1 The *naive* assumption: conditional on the latent variables U_i, U_j, all relations defining $\omega \downarrow (i, j)$ are independent (in the spirit of the naive Bayes assumption).

A2 The *histogram* assumption: $f_2^<$ is piecewise constant on a grid partition of its domain $[0, 1] \times [0, 1]$, i.e., $f_2^<(u, u') = f_2^<(b(u), b(u'))$ where $b(u)$ is the bin index of u in a fixed given partioning of $[0, 1]$ into interval bins $b_1, \ldots, b_g$ with a *granularity* parameter $g \in \mathbb{N}$.

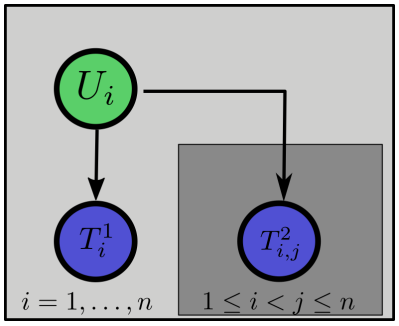


Figure 1: Plate representation of AHK$^-$ model

Under these assumptions, $f_2^<$ can be represented as a $g \times g \times |\mathcal{E}| \times 2$-dimensional array F_2, where $F_2[k, h, l, 0]$ and $F_2[k, h, l, 1]$ specify the probabilities of an edge $E_l(i, j)$, respectively $E_l(j, i)$, given that $u_i < u_j$, $k = b(u_i)$, and $h = b(u_j)$. We can then substitute in (5):

$$f_2^<(u_p, u_q)(\omega \downarrow (\pi(p), \pi(q))) =$$
$$\prod_{l=1}^{|\mathcal{E}|} \prod_{d \in \{0,1\}} \boldsymbol{E}[\pi(p), \pi(q), l] \cdot F_2[b_p, b_q, l, d] + (1 - \boldsymbol{E}[\pi(p), \pi(q), l] \cdot (1 - F_2[b_p, b_q, l, d]), \quad (6)$$

where $\boldsymbol{E}$ is the edge relation tensor of ω. The right-hand side here depends on π and $\boldsymbol{u}$ only through the relative order and bin membership of the two nodes $\pi(p), \pi(q)$. The same term will be encountered in many evaluations of $\pi, \boldsymbol{u}$ pairs of the outer summation/integration in (5). We collect these values in a $g \times g \times n \times n$-dimensional table BP_ω, so that $BP_\omega(k, h, i, j)$ contains the multiplicative factor contributed to $P(\omega)$ by the nodes i, j, whenever $b(u_i) = k, b(u_j) = h$, and $\pi^{-1}(i) < \pi^{-1}(j)$. Replacing the integral over $\boldsymbol{u}$ by a sum over possible induced bin membership vectors, then (5) can be written as (but still omitting f_1):

$$P(\omega) = \sum_{\pi \in \mathcal{P}[n]} \sum_{\boldsymbol{b} \in \langle g \rangle^n} \prod_{(p,q) \in \langle n \rangle^2} BP_\omega(b_p, b_q, \pi(p), \pi(q))V(\boldsymbol{b}), \quad (7)$$

where $V(\boldsymbol{b})$ is the volume of the n-dimensional hypercube defined by the bin sequence $\boldsymbol{b}$, intersected with $\langle[0, 1]\rangle^n$.

All definitions here are given for directed graphs. Undirected graphs are handled by a slightly simplified model where the table F_2 only needs to be $|\mathcal{E}| \times g \times g$-dimensional, omitting the argument that encodes the direction of an edge. The probability of an incomplete graph is computed in exactly the same way by taking the product in (6) only over the relations A_l for which $\boldsymbol{E}(\pi(p), \pi(q), l) \neq ?$. Importantly, due to **A1**, no summation over possible values of unobserved edges or attributes is needed.

5 Expressivity and Complexity

The general AHK$^-$ model generalizes the basic graphon model with the ability to model discrete node attributes and multiple directed edge relations. The NH-AHK$^-$ model with its restriction to histogram

representations generalizes the stochastic block model in a similar manner. Though NH-AHK$^-$ is conceived as a lightweight alternative to more expressive SRL languages, its latent variables enable some modeling capabilities not commonly found in existing SRL frameworks. An example of this will be given in Section 7.1.

While the shared nature of being generative random graph models supports a more direct expressivity comparison between NH-AHK$^-$ and stochastic block models or SRL frameworks, one can also compare the capabilities of NH-AHK$^-$ vs. graph neural networks in terms of discriminating different structures. Figure 2 shows a standard example illustrating limitations of graph neural networks: the two graphs and their nodes are indistinguishable by standard GNN architectures in the sense that every such GNN will construct node or graph feature vectors that are identical for the two graphs [1]. A stochastic block model (and hence a NH-AHK$^-$ model) with two blocks and high intra-block and low inter-block probabilities, on the other hand, will assign a much higher probability to the right than the left graph, and thus distinguish these two graphs.

A fundamental limitation of the AHK model is the inability to represent sparse random graph models: the expected number of edges necessarily grows linearly in the number of possible edges, i.e., quadratic in the number of nodes.

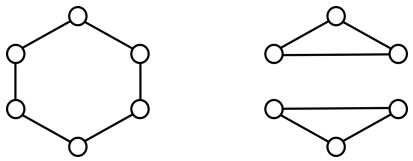


Figure 2: GNN indistinguishable graphs

Even though NH-AHK$^-$ makes the significant simplifying assumptions A1 and A2, it still leads to high complexity of probabilistic inference. A direct evaluation of (7) is exponential in n, and hence intractable for all but very small graphs. As the following proposition shows, this is an inherent complexity that is unlikely to be avoidable by other inference designs.

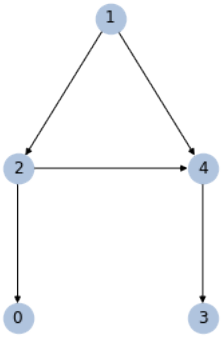
Proposition 5.1 Computing the probability $P(\omega)$ of a graph ω in an NH-AHK$^-$ model is NP-hard.

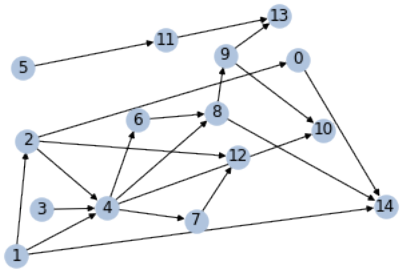
The proof is by an easy reduction of the 3-colorability problem: consider an NH-AHK$^-$ model that represents a simple stochastic block model with 3 blocks, where intra-block links have probability zero, and inter-block links have probabilities > 0. Then a graph has a nonzero probability under this model, iff it is 3-colorable.

6 Inference and Learning

For inference we need to evaluate (7). For learning we need to compute the gradient of (7) with respect to the parameters contained in the array F_2. Given the gradients, and assuming a fixed bin partitioning, learning is performed by stochastic gradient descent using Adam [15]. If the granularity is given, but the boundary points defining the bins are to be optimized, then the gradient of (7) also needs to be computed with respect to these boundary points (which affect (7) only via the volume factors $V(\boldsymbol{b})$). The granularity itself can either be viewed as a hyper-parameter that can be set or optimized using the common approaches to hyperparameter tuning, or one can use an iterative refinement approach where learning starts at $g = 1$, and when learning at a given g value has terminated, then the currently widest bin is split into two, and learning continues at $g + 1$. This is iterated until the log-likelihood obtained at a g level does not exceed the likelihood at $g - 1$ by more than a specified factor. Key to making both inference and learning tractabel is to replace the outer two sums over $\pi \in \mathcal{P}[n]$ and $\boldsymbol{b} \in \langle g \rangle^n$ by an expectation over sampling with a proposal distribution $Q(\pi, \boldsymbol{b})$, such that $Q(\pi, \boldsymbol{b})$ approximates the posterior $P(\pi, \boldsymbol{b}|\omega) \sim P(\omega|\pi, \boldsymbol{b})V(\boldsymbol{b})$ (from Bayes's rule, and using that $P(\pi, \boldsymbol{b}) = n!V(\boldsymbol{b})$ is constant in π). We sample $\pi, \boldsymbol{b}$ by iteratively inserting a next node into already partially constructed π and $\boldsymbol{b}$ tuples. We make the probability of inserting the next node i at a given position with a given bin assignment proportional to the product of:

- the BP_ω values we obtain from pairs i, j for nodes j already contained in the current partial π. Note that inserting node i into π does not change any of the BP_ω factors contributed by pairs of nodes j, j' already contained in π, since these values only depend on the relative order of j and j' and their bin memberships, but not on their absolute positions in π.





Query edge(s)	Probability
$1 \to 0$	0.43
$0 \to 3$	0.29
$3 \to 0$	0.16
$3 \to 2$	0.005
$1 \to 0 \& 0 \to 3$	0.134
$0 \to 3 \& 3 \to 0$	0.0017

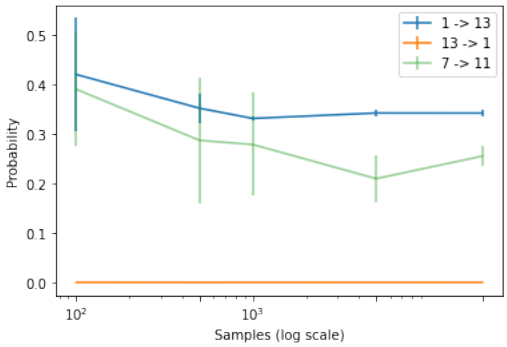


Figure 3: Queries on DAG model

- an estimate of the BP_ω values we will obtain from pairs i, j for nodes j that are not yet contained in pi. For this we take a maximum over the BP_ω values that can be obtained by a future insertion of j before or after the position of i with the corresponding possible bin assignments for j.

The basic idea can be summarized as replacing an exact optimization over $n!$ many possible permutations π by an approximation that only takes pairwise orderings of elements in π into account. A detailed description of the samping algorithm and an illustration of its effectiveness is contained in Appendix A.

Drawing a single sample from Q still is cubic in n (see Appendix A). This implies strict limitations on the sizes of graphs that can be handled. In the case of probabilistic inference, these limitations are often not so problematic, since queries often only refer to a small number of nodes of interest. Training data, on the other hand, will often consist of graphs that are intractable for approximate gradient computations. In this case, we use a subsampling strategy, and learn from small induced subgraphs obtained by random uniform sampling of nodes in the original training graphs.

7 Examples and Experiments

7.1 Directed Acyclic Graphs

In this section we demonstrate the capabilities of the NH-AHK$^-$ model by learning and reasoning with directed acyclic graphs. We created a training set containing 100 small directed acyclic graphs with 3 to 7 nodes. The graphs are directed Erdős-Rényi random graphs filtered on being acyclic. The sampled graphs had an average edge density of 0.43 (w.r.t. to the maximal number of $n \cdot (n-1)/2$ edges in an acyclic graph).

We apply the learned model to answer several queries about the graphs shown in Figure 3. The edges shown in the graphs are known to exist. All other edges are considered unknown, and we query the probability of their existence. The first three queries for the graph on the left are about edges that are consistent with acyclicity and the observed edges. Their probabilities depend on the number of complete orderings of the nodes that are consistent with the partial order imposed by

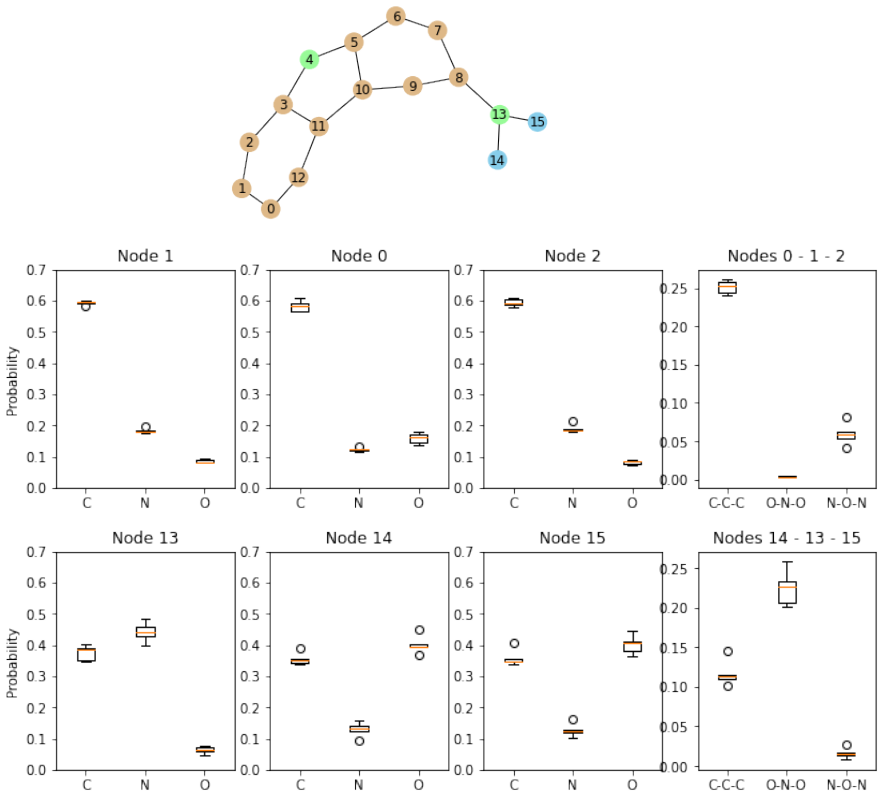


Figure 4: Top: Query molecule; brown: carbon (C), green: nitrogen (N), blue: oxygen (O). Box plots of computed probabilities for single node and multiple-node queries.

the observed edges. The edge $1 \to 0$ is consistent with every possible complete ordering, and its probability is the overall edge density in the training set. The relative order between nodes 0 and 3 is not fully determined by the observed edges, but there are more complete orderings in which 0 preceds 3 than vice-versa, which is why the edge $0 \to 3$ has a higher probability. The edge $3 \to 2$ is inconsistent with acyclicity, and therefore has a very low probability. The final two queries are about the probabilities of joint occurrences of two edges. The first combination of edges is still consistent with acyclicity, and the probability for the combination is close to the product of the individual edge probabilities. The last combination introduces a cycle, and therefore it has a much lower probability than the product of individual edge probabilities.

Inference for the small graph on the left can be performed exactly. Figure 3 on the right shows a larger query graph for which approximate inference has to be used. This graph also is twice as big as the largest training graph. We query the model for the probabilities of the edges $1 \to 13$, $13 \to 1$, and $7 \to 11$. The plot at the bottom right shows for $N = 100, 500, 1000, 5000, 20000$ samples the computed probabilities for the three queries. The plots show average and standard deviation over 5 inference runs. We observe that at $N = 5000$ the approximate inference results are already quite stable. The estimated probabilities obtained at $N = 20000$ are 0.342, 0.001, and 0.255, respectively, for the three queries, again reflecting the proportion of total node orderings that are consistent with the given partial topological order and the query edge direction. The time for computing a single sample was approximately 0.006 seconds, or 30s for one inference run at $N = 5000$ (jointly for all 3 queries).

7.2 Molecules

The well-known MUTAG dataset consists of 188 molecules of 10-28 atoms (nodes). Nodes have an attribute *element* $\in \{C, N, O, F, I, Cl, Br\}$, and molecules carry a Boolean class label *mutagenic*. The standard task is to predict the class labels. However, we here use this dataset to learn a model for

probabilistic inference. To this end, we use the first 187 molecules for training, and consider the last molecule as a subject for inference. Due to the small size of the molecules, no sub-sampling of the training graphs here is necessary. We learn with a fixed granularity of $g = 10$.

Figure 4 at the top shows the query molecule. Like most molecules in the dataset, it only contains the elements C,N and O. We consider atoms 0,1,2 and 13,14,15 as query atoms and set their element attribute to unknown. Given the known graph structure (different from Figure 3, edges not shown in Figure 4 are known not to exist) and the observed elements of the atoms 3,. . . ,12, we want to infer the elements of the query atoms. The trained NH-AHK$^-$ model allows us to both query for the marginal probabilities of each query atom individually, and to query for the probabilities of joint configurations of multiple query atoms. We pose these joint queries for the two groups of query atoms, computing the probabilities of all joint configurations that involve the three elements C,N,O, and in which the two atoms 0,2, respectively 14,15, are the same element. We perform importance sampling inference with $N = 5000$ samples, repeated for 5 runs. Figure 4 shows boxplots of the obtained probability estimates in the 5 runs. For nodes 0,1,2 we obtain for the single node probabilities a high probability for C, and much lower probabilities for N and O (top 3 left plots). This is quite in line with the result for the joint configuration, where the C-C-C configuration is about five times as likely as the second most probable N-O-N configuration (top right plot; joint configurations not shown in the boxplots all have very low probabilities). For the 13,14,15 group we observe in the single node inferences that N for 13 and O of 14,15 are the most probable elements. However, in all cases C is almost as likely. Here joint inference provides a quite different picture in that now the joint configuration O-N-O is about twice as likely as the C-C-C configuration. This example demonstrates how the NH-AHK$^-$ model can model mutual dependencies that can be exploited for better predictions of joint configurations than what can be obtained by just combining independent individual predictions (in other cases it may very well be the case that the most probable joint configuration is not the combination of the most probable individual attributes).

7.3 Graph Generation

The NH-AHK$^-$ model allows efficient sampling of graphs of arbitrary size. In this section we explore the capability of NH-AHK$^-$ to serve as a simple baseline approach to learning graph generators. Observe that one must distinguish graph generators and generative models: the former are designed only to support sampling of random graphs; the latter are defined by the semantic property of specifying a full joint distribution, and are traditionally intended to support probabilistic and predictive inference. Dedicated graph generators usually do not support any inference, and not every generative model supports efficient sampling from the distribution it defines.

Datasets and settings: We evaluate our model using two well-known graph benchmarks: *Community* [27] and *EGO* [27] networks. The *Community* dataset consists of 500 synthetic two-community graphs. *Community* fits the modeling capabilities of the NH-AHK$^-$ model very well. The *EGO* dataset comprises 816 2-hop ego networks extracted from the Citeseer network [22]. The sizes of the graphs range between $4 \leq n \leq 262$. *EGO* appears like a reasonable challenge for NH-AHK$^-$. We note that other common benchmarks like *Grid* [27] represent sparse graph distributions that are inherently out of scope for NH-AHK$^-$.

Our primary focus is on a generator's extrapolation capability: to generate larger graphs than seen in the training data. We therefore use separate test sets that contain graphs of the same size as the training graphs, and graphs that are (on average) larger by scaling factors (sf) of $1.5, 2, 4$, and 8. For *Community* we just create synthetic examples of the appropriate sizes. The *EGO* graphs are partitioned into subsets according to their size, such that the different sets differ (approximately) by our scaling factors. The training set then contains graphs of sizes $4 \leq n \leq 18$.

Other approaches: We compare NH-AHK$^-$ against two popular architectures for graph generation: GraphRNN [27] generates new graphs in an incremental, auto-regressive manner. DiGress [26] is a diffusion-based, one-shot generative model based on graph transformers. We have selected these two models because of their complementary designs, and because they are able, in principle, to generate graphs of arbitrary size. As a very simple baseline we include the Erdős–Rényi (*ER*) model [8].

Metrics: A common score to evaluate graph generators is the maximum mean discrepancy (MMD) of degree, clustering, and orbit distributions[17, 19, 26, 27]. However, it has recently been shown that MMD is quite sensitive to the choice its parameters [20]. Moreover, degree, clustering, and orbit distribution do not capture well the characteristic structure of *Community* and *EGO* networks.

Method	*Community*						*EGO*					
	Stat.	Scaling factor					Stat.	Scaling factor				
		1	1.5	2	4	8		1	1.5	2	4	8
NH-AHK$^-$	nb of com.	0.02	**0.00**	**0.00**	**0.00**	**0.00**	dia.	0.47	**0.75**	1.00	1.15	1.73
DiGress		**0.00**	0.20	0.53	-	-		**0.32**	0.78	**0.33**	**0.35**	**0.40**
GraphRNN		1.05	-	-	-	-		2.08	-	-	-	-
ER		2.42	2.07	1.90	1.25	1.23		0.40	2.03	1.48	0.96	2.05
NH-AHK$^-$	mod.	0.03	**0.03**	**0.02**	**0.02**	**0.02**	rad.	0.39	**0.13**	**0.03**	**0.10**	**0.05**
DiGress		**0.00**	0.08	0.14	-	-		**0.08**	0.29	0.05	0.85	1.05
GraphRNN		0.06	-	-	-	-		1.11	-	-	-	-
ER		0.29	0.34	0.36	0.40	0.43		0.39	1.13	1.29	1.77	1.94

Table 1: Community and ego network results: EMD between number of communities (top-left), modularity (bottom-left), diameter (top-right) and radius (bottom-right) of generated and test networks, for increasing values of the scaling factor between training and generated/test networks. Best results are highlighted in bold.

We therefore introduce metrics that are more tailored towards these particular benchmarks. For *Community* we measure similarity between generated and test set by comparing the distribution of the number of communities and their modularity scores [1]. A characteristic feature of ego-networks is the distribution of pair-wise distances between nodes, which (for our 2-hop networks) is bounded by 4, and, for at least one node, all distances to other nodes are bounded by 2. We use the radius and the diameter to capture essential distance properties of a graph. All our base metrics, thus, are graph-level measures, and we compare two graph sets by the earth-movers distance (EMD) between the distributions of these measures. In Appendix B.5 we also report results using the standard MMD metrics.

Results: In Table 1, we present the Earth Mover's Distance (EMD) between the generated and test networks for each scaling factor for *Community* and *EGO* networks, left and right of Table 1, respectively. A scaling factor equal to 1 indicates that the generated networks are of the same size as the training set. Notably, certain entries of *GraphRNN* are left unfilled due to the models' inability to generate larger networks (for further details, refer to Appendix B.4). While *DiGress* encounters memory limitations when generating *Community* networks with a scaling factor of 4 and 8. Examining the *Community* results, it is evident that despite competitors being tailored specifically for graph generation, NH-AHK$^-$ demonstrates comparable results in network generation and surpasses them in generating larger networks. Furthermore, it is important to note that the NH-AHK$^-$ model for this dataset required only 3 parameters, while *GraphRNN* and *DiGress* require almost 400,000 and 4.6 million parameters, respectively. On the other hand, within the *EGO* networks, the NH-AHK$^-$ model can capture the radius but fails in capturing the diameter. Once again, the main objective here is to demonstrate that the NH-AHK$^-$ model, with only 6 parameters, stands in comparison with specific graph generation models: *GraphRNN* with nearly 400,000 parameters and *DiGress* with 2 million parameters. We provide a visualization of generated graphs in Appendix B.6.

8 Conclusion

In this paper we have introduced theory and implementation of the NH-AHK$^-$ model. The model is conceived as a highly versatile, lightweight graph learning and inference tool that combines key features of SRL frameworks on the one hand, and GNNs on the other. Like SRL models it is fully generative and supports a wide range of probabilistic queries beyond simple classification tasks. We have demonstrated the capabilities of answering joint queries for multiple edges, or multiple node attributes. Like GNNs, NH-AHK$^-$ does not rely on expert knowledge to specify the logical structure of a model in some form of probabilistic programming language. For the purpose of graph generation we found that NH-AHK$^-$ can compete with, and partly outperform highly engineered specialized GNN models on tasks that are compatible with the inherently dense nature of NH-AHK$^-$ models. Other advantages of NH-AHK$^-$ beyond what has been demonstrated in this paper include the support for multi-relational graphs, and the ability to learn from incomplete data.

[1] computed with the Clauset-Newman-Moore greedy modularity maximization algorithm [6]

Acknowledgments

This research was supported by TAILOR, a project funded by the EU Horizon 2020 research and innovation program under GA No 952215. AL acknowledges the support of the MUR PNRR project FAIR - Future AI Research (PE00000013) funded by the NextGenerationEU.

References

[1] Ralph Abboud, Ismail Ilkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization. In *Proceedings of IJCAI 2021*, 2021. 5

[2] Edo M Airoldi, Thiago B Costa, and Stanley H Chan. Stochastic blockmodel approximation of a graphon: Theory and consistent estimation. *Advances in Neural Information Processing Systems*, 26, 2013. 2

[3] David J Aldous. Representations for partially exchangeable arrays of random variables. *Journal of Multivariate Analysis*, 11(4):581–598, 1981. 1

[4] Stanley Chan and Edoardo Airoldi. A consistent histogram estimator for exchangeable graph models. In *International Conference on Machine Learning*, pages 208–216. PMLR, 2014. 2

[5] Swati Chandna, SC Olhede, and PJ Wolfe. Local linear graphon estimation using covariates. *Biometrika*, 109(3):721–734, 2022. 2

[6] Aaron Clauset, Mark EJ Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical review E*, 70(6):066111, 2004. 9, 17

[7] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, page 2468–2473, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc. 1

[8] Paul Erdős, Alfréd Rényi, et al. On the evolution of random graphs. *Publ. math. inst. hung. acad. sci*, 5(1):17–60, 1960. 8

[9] Chao Gao, Yu Lu, and Harrison H Zhou. Rate-optimal graphon estimation. *The Annals of Statistics*, pages 2624–2652, 2015. 2

[10] Douglas N. Hoover. Relations on probability spaces and arrays of random variables. *HPreprint, Institute for Advanced Study, Princeton, NJ*, 2, 1979. 1

[11] M. Jaeger and G. Van den Broeck. Liftability of probabilistic inference: Upper and lower bounds. In *Proceedings of the 2nd International Workshop on Statistical Relational AI*, 2012. 1

[12] Manfred Jaeger. Relational bayesian networks. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, UAI'97, page 266–273, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc. ISBN 1558604855. 1

[13] Manfred Jaeger and Oliver Schulte. A complete characterization of projectivity for statistical relational models. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI)*, 2020. 1, 3

[14] Olav Kallenberg. *Probabilistic symmetries and invariance principles*. Springer Science & Business Media, 2006. 1

[15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 5

[16] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009. 1

[17] Jenny Liu, Aviral Kumar, Jimmy Ba, Jamie Kiros, and Kevin Swersky. Graph normalizing flows. *Advances in Neural Information Processing Systems*, 32, 2019. 9

[18] James Lloyd, Peter Orbanz, Zoubin Ghahramani, and Daniel M Roy. Random function priors for exchangeable arrays with applications to graphs and relational data. *Advances in Neural Information Processing Systems*, 25, 2012. 2

[19] Chenhao Niu, Yang Song, Jiaming Song, Shengjia Zhao, Aditya Grover, and Stefano Ermon. Permutation invariant graph generation via score-based generative modeling. In *International Conference on Artificial Intelligence and Statistics*, pages 4474–4484. PMLR, 2020. 9

[20] Leslie O'Bray, Max Horn, Bastian Rieck, and Karsten Borgwardt. Evaluation metrics for graph generative models: Problems, pitfalls, and practical solutions. *arXiv preprint arXiv:2106.01098*, 2021. 9

[21] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine Learning*, 62 (1-2):107–136, 2006. 1

[22] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008. 8

[23] Cosma Rohilla Shalizi and Alessandro Rinaldo. Consistency under sampling of exponential random graph models. *Annals of statistics*, 41(2):508, 2013. 1

[24] Benjamin Sischka and Göran Kauermann. Em-based smooth graphon estimation using mcmc and spline-based approaches. *Social Networks*, 68:279–295, 2022. 2

[25] Yi Su, Raymond KW Wong, and Thomas CM Lee. Network estimation via graphon with node features. *IEEE Transactions on Network Science and Engineering*, 7(3):2078–2089, 2020. 2

[26] Clement Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. Digress: Discrete denoising diffusion for graph generation. *arXiv preprint arXiv:2209.14734*, 2022. 8, 9, 14, 15

[27] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International conference on machine learning*, pages 5708–5717. PMLR, 2018. 8, 9, 14, 15

A Importance Sampling

Given an NH-AHK$^-$ model and a graph ω, the goal is to sample a random pair of a permutation π and a bin assignment $\boldsymbol{b}$, such that the sampling probability $Q(\pi, \boldsymbol{b})$ approximates the posterior distribution $P(\pi, \boldsymbol{b}|\omega)$. For a full description of the method, we need to also provide the details of the handling of the f_1 factors in (5), which in (6) and (7) were omitted. Without loss of generality, we assume that categorical attribute A_l has values in $[m_l]$ for some $m_l \in \mathbb{N}$. Under assumptions **A1**,**A2**, the function f_1 can be represented by $g \times m_h$-dimensional tables $F_{1,l}$ ($l = 1, \ldots, |\mathcal{A}|$), such that $F_{1,l}[k, v]$ is the probability that a node i with $k = b(u_i)$ has value v for attribute A_l (i.e., the rows of $F_{1,l}$ are multinomial distributions over the values of A_l).

For a given graph ω, we define in analogy to BP_ω:

$$UP_\omega(k, i) = \prod_{l=1}^{|\mathcal{A}|} \sum_{v \in [m_l]} \mathbb{I}[\boldsymbol{A}[i, l] = v] F_{1,l}[k, v], \tag{8}$$

where $\mathbb{I}[]$ is the indicator function.

ISAMPLE (n,g,BP,UP)

π_{init}= random permutation of $[n]$
$\pi = [], \boldsymbol{b} = []$ `/* Initialize as empty lists */`

for i=1,...,n **do**
  next=$\pi_{init}[i]$
  remaining=$\pi_{init}[i+1:n]$
  insertions=list of possible combinations of insertion position for *next* into π, and associated bin assignement.
  sampleprobs=GETSAMPLEPROBS(*insertions*,π,$\boldsymbol{b}$,*remaining*,BP,UP)
  sample an insertion position/bin according to *sampleprobs*
  insert *next* at the sampled position/bin
end
return $\pi, \boldsymbol{b}$

Algorithm 1: High-level importance sampling algorithm

The importance sampling algorithm receives as input the pre-computed tables BP_ω, UP_ω. The high-level structure of the algorithm is shown in Algorithm 1. We start by creating an initial random

permutation π_{init} of the nodes, and then take nodes in the order of π_{init} to iteratively insert them into the final permutation π. Given partially constructed $\pi, \boldsymbol{b}$, the list of possible combinations of insertion positions and bin assignments is computed in line 7. For example, if $n = 5, g = 3$ and currently $\pi = (4, 1), \boldsymbol{b} = (1, 2)$, then *insertions* $= ((0, 0), (0, 1), (1, 1), (1, 2), (2, 2)$, meaning that the next node can be inserted as the first element of π with bin assignment 0 or 1, in the middle between nodes 4 and 1 with bin assignment 1 or 2, or at the end with bin assignment 2 (this only depends on g and the current $\boldsymbol{b}$, not the current π). The core of the importance sampling procedure then consists of line 8 where a probability distribution over *insertions* is calculated by Algorithm 7.

GETSAMPLEPROBS (*insertions*,π,$\boldsymbol{b}$,*next*,*remaining*,*BP*,*UP*)

1 *weights*=[]
2 **for** $(pos, bin) \in insertions$ **do**
3 $\quad w =$ *leftweight*$(next, \pi, \boldsymbol{b}, pos, bin) \cdot$ *rightweight*$(next, \pi, \boldsymbol{b}, pos, bin) \cdot$ *remainweight*$(next, remaining, bin)$
4 $\quad w = w \cdot UP[bin, next]$
5 $\quad$ append w to *weights*
6 **end**
7 **return** weights *normalized to probability distribution*

Algorithm 2: Core weighting heuristic

The main component of this algorithm is line 3, where for a candidate insertion (pos, bin) three weight factors are combined: *leftweight* is the contribution to $P(\omega|\pi, \boldsymbol{b})$ (for the final $\pi, \boldsymbol{b}$, currently under construction) that derives from node pairs $i, next$ for nodes i that are already included in the current π to the left of the candidate insertion position *pos*:

$$leftweight(next, \pi, \boldsymbol{b}, pos, bin) := \prod_{p=0}^{pos-1} BP[b_p, bin, \pi[p], next] \tag{9}$$

In our previous example: if $next = 3$, and $(pos, bin) = (1, 2) \in insertions$, then $leftweight(3, (4, 1), (1, 2), 1, 2) = BP[1, 2, 4, 3]$. Analogously, we define:

$$rightweight(next, \pi, \boldsymbol{b}, pos, bin) := \prod_{p=pos+1}^{length(\pi)-1} BP[bin, b_p, next, \pi[p]] \tag{10}$$

In our example: $rightweight(3, (4, 1), (1, 2), 1, 2) = BP[3, 1, 2, 2]$. The *remainweight* term estimates the contribution to $P(\omega|\pi, \boldsymbol{b})$ from node pairs $i, next$ for nodes $i \in remaining$. Since positions and bin memberships of these i are not yet determined, we make an optimistic estimate by maximizing over all possibilities of placing i before or after *next*, and the corresponding possible bin assignments for i:

$$remainweight(next, remaining, pos, bin) := \prod_{i \in remaining} \max\{\max_{b \le bin} BP[b, bin, i, next], \max_{b \ge bin} BP[bin, b, next, i]\} \tag{11}$$

In our example, let $remaining = [0, 2]$. Then $remainweight(3, [0, 2], 2) = \prod_{i \in \{0,2\}} \max\{BP[0, 2, i, 3], BP[1, 2, i, 3], BP[2, 2, i, 3], BP[2, 2, 3, i]\}$

Line 4 multiplies the weight computed so far with the factor $UP[bin, next]$ that represents the contribution to $P(\omega|\pi, \boldsymbol{b})$ that derives from the probabilities of the node attributes. This only depends on the bin membership of *next*, not the permutation π.

Complexity. (Sketch) The total number of insertions calculated in line 7 of ISAMPLE is $O(n(n+g))$ (over all iterations of the for-loop). The weight computation for one candidate insertion in lines 3-5 of GETSAMPLEPROBS is $O(ng)$ (the combined computation of *leftweight*,*rightweight*, *remainweight* effectively iterates over all nodes other than *next*, and the computation of *remainweight* also involves an iteration over the bins). This gives a total complexity of $O(n^3 g + n^2 g^2)$, which also dominates the complexity of the pre-computation of *BP*, which is $O(n^2 g^2)$.

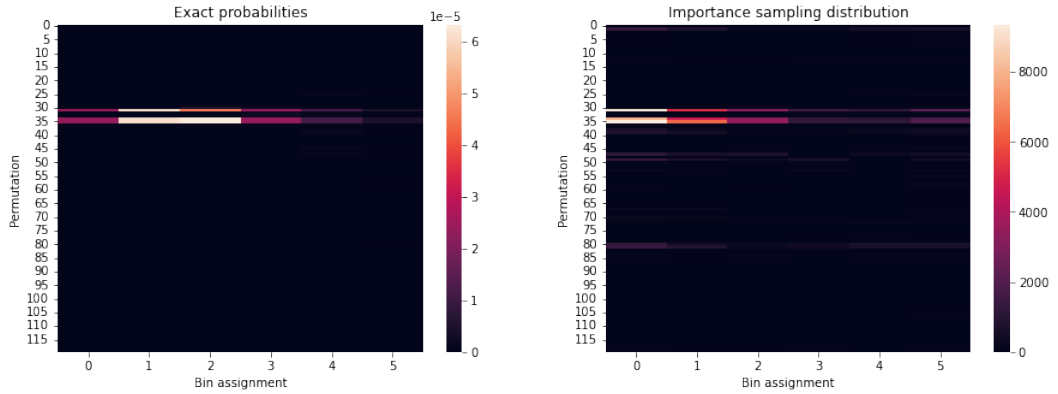


Figure 5: Exact and importance sampling distribution for the DAG of Figure3

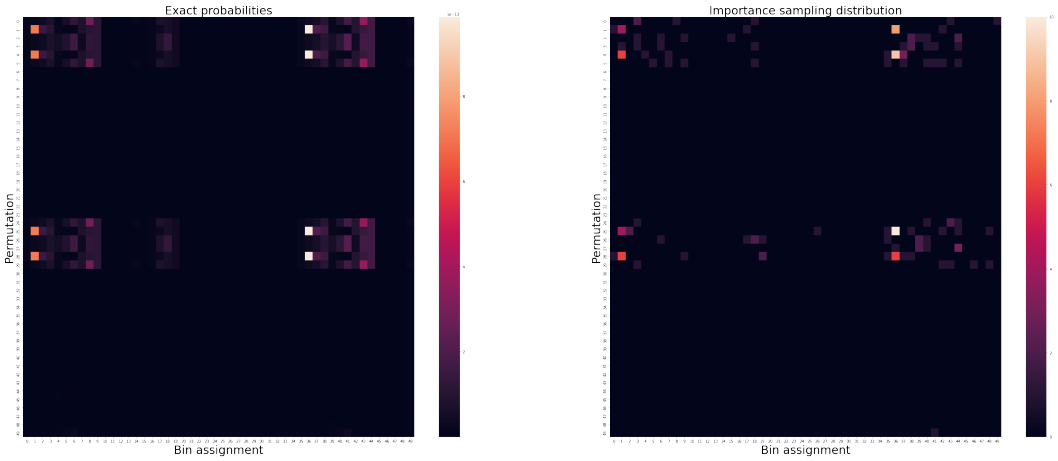


Figure 6: Exact and importance sampling distribution for the ego graph of radius 2 of node 13 in Figure 4

Illustration. The effectiveness of the importance sampling method is demonstrated by the following comparison of the actual probabilities $P(\omega, \pi, \boldsymbol{b})$, and empirical sample probabilities obtained from ISAMPLE. Figure 5 shows heatmap representations of the values of $P(\omega, \pi, \boldsymbol{b})$ (left) and the empirical distribution of 100.000 samples from ISAMPLE. Here ω is the 5 node query graph shown on the left of Figure 3, and the underlying model is the model learned in Section 7.1, which has a granularity $g = 2$. Thus, there are $5! = 120$ permutations (y-axes in the figure), and $\binom{5+2-1}{5} = 6$ different $\boldsymbol{b}$ vectors (x-axes in the figure). $P(\omega, \pi, \boldsymbol{b})$ is sharply concentrated on the three permutations that are topological orders of the graph, and so is the sampling distribution.

Figure 6 gives a similar illustration for the MUTAG example of Section 7.2. We use as ω the ego-graph of radius 2 of node 13, which consists of 6 nodes. With a model granularity of $g = 10$, this gives us $6! = 720$ permutations and $\binom{6+10-1}{6} = 5005$ different $\boldsymbol{b}$ vectors. Figure 6 displays a detail of size 50×50 of the resulting 720×5005-dimensional heatmaps. Again we observe a very good match between the true and the sampling distribution.

B Generated graph evaluation

B.1 Graph Generation Baselines

The official repository of *GraphRNN* [27] and *DiGress* [26] were used for implementing the two baselines in Section 7.3. For *GraphRNN* we stuck to the original configuration provided in the repository for all datasets, while for *DiGress* on the *EGO* dataset we reduced the number of layers to

5 to account for the smaller size of the networks. For model selection, we resorted to the heuristics already provided in the repositories, selecting as the best epoch for *DiGress* over *EGO* and *Community* respectively epoch 799 and 699, while for *GraphRNN* 1600 and 1600. The experiments were launched on a single A100 80GB GPU.

B.2 Data Details

Community: Each community is generated by the Erdos-Renyi model with $N \sim U(30, 80)$ many nodes, and intra-community edge probability $p = 0.3$. The communities are connected by random edges with edge probability of 0.0025.

EGO: The collection of ego-graphs is divided as follows:

Scaling factor	Number training graphs	Number test graphs	Size range
1.0	94	24	4-18
1.5	-	227	19-24
2.0	-	134	25-33
4.0	-	198	34-67
8.0	-	139	70-262

B.3 Learned Models

The learned NH-AHK$^-$ model for the *EGO* data is defined by the binbounds $0, 0.25, 0.5, 1.0$ and

$$F_2 = \begin{pmatrix} 0.24311648 & 0.87151754 & 0.68273159 \\ & 0.01519822 & 0.05864193 \\ & & 0.15766381 \end{pmatrix}$$

$F_2[k, h]$ is the probability of an edge between nodes belonging to bins k and h (the graphs here are undirected, so that only the upper triangular matrix is needed) According to this model, nodes belonging to the first bin $[0, 0.25]$ are highly connected to nodes in the other two bins; nodes in the second bin are sparsely connected to nodes in any other bin than the first, and nodes in the third bin also have a certain edge density (0.157) among themselves, apart from having a high connectivity with the nodes in the first bin.

The learned NH-AHK$^-$ model for the *Community* data is defined by the binbounds $0, 0.5, 1.0$ and

$$F_2 = \begin{pmatrix} 0.30327248 & 0.00788412 \\ & 0.25864134 \end{pmatrix}$$

which corresponds to a simple stochastic block model for two communities.

B.4 Generating Bigger Graphs

While the number of edges is usually upper-bounded by the maximum size of the BFS queue found in the train set, *GraphRNN* [27] can generate an arbitrary number of nodes by unrolling the trained RNN. *DiGress* [26], on the other hand, can generate bigger graphs by sampling the number of nodes from a desired node distribution. In Table 2 we reported the average number of nodes for the reference test set and the generated graphs. The results show an inherent limitation of *GraphRNN* in generating larger networks, in contrast to all the other methods. We claim that this behavior can be due to the local auto-regressive nature of the architecture, which seems to suffer from overfitting on the training distribution, which in turn generally shows a pretty narrow variability in the number of nodes. The same does not happen for *DiGress*, which first samples a noisy graph of size *n* and then reverts the noise via the trained Graph Transformer [26].

B.5 MMD Evaluation

To ensure a rigorous evaluation, we assess the performance of the NH-AHK$^-$ model using the Maximum Mean Discrepancy (MMD) metric across degree distributions, clustering distributions, and orbit distributions. We compute these statistical measures and the MMD score employing the same framework as utilized by *GraphRNN*. The corresponding source code can be accessed at the following link[2].

[2]https://github.com/snap-stanford/GraphRNN

Scaling Factor	Test set	NH-AHK$^-$	*GraphRNN*	*DiGress*	*ER*
1.0	8.9 (±4.1)	9.7 (±3.7)	14.8 (±1.3)	7.5 (±4.7)	8.9 (±4.1)
1.5	20.3 (±2.0)	20.6 (±1.8)	15.0 (±1.1)	14.6 (±4.9)	20.3 (±2.0)
2.0	29.4 (±2.7)	28.5 (±2.9)	14.9 (±1.5)	24.4 (±3.7)	29.4 (±2.7)
4.0	45.4 (±8.8)	45.8 (±10.2)	15.0 (±1.5)	57.3 (±3.6)	45.4 (±8.8)
8.0	118.8 (±27.8)	115.0 (±20.9)	14.8 (±1.4)	127.1 (±3.3)	118.8 (±27.8)

(a) Average number of nodes of EGO networks

Scaling Factor	Test set	NH-AHK$^-$	*GraphRNN*	*DiGress*	*ER*
1.0	98.4 (±28.8)	102.8 (±26.0)	157.4 (±2.2)	110.8 (±30.2)	98.4 (±28.8)
1.5	167.3 (±42.2)	164.2 (±43.8)	158.8 (±2.4)	173.9 (±26.3)	167.3 (±42.2)
2.0	235.6 (±55.9)	217.3 (±61.2)	160.3 (±1.1)	264.5 (±26.1)	235.6 (±55.9)
4.0	445.4 (±117.3)	446.6 (±120.3)	160.8 (±2.8)	-	445.4 (±117.3)
8.0	889.5 (±198.2)	910.4 (±209.5)	165.6 (±2.1)	-	889.5 (±198.2)

(b) Average number of nodes of Comm networks

Table 2: Average number of nodes and standard deviation for reference test sets of *EGO* and *Community*, and of the generated graphs

		EGO					*Community*				
Scaling Factor		1.0	1.5	2.0	4.0	8.0	1.0	1.5	2.0	4.0	8.0
deg.	NH-AHK$^-$	0.17	0.73	0.78	0.58	**0.46**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
	DiGress	**0.07**	**0.06**	0.59	1.09	1.12	**0.00**	0.01	0.01	-	-
	GraphRNN	0.02					0.15				
	ER	0.09	0.33	**0.50**	**0.47**	0.89	0.01	0.01	0.01	0.01	0.01
clust.	NH-AHK$^-$	0.03	0.24	0.23	0.25	0.17	0.06	0.07	0.05	0.09	0.06
	digress	0.04	**0.01**	0.08	0.17	0.20	0.06	0.12	0.23	-	-
	GraphRNN	0.04					**0.01**				
	ER	**0.02**	0.03	**0.05**	**0.07**	**0.12**	0.02	**0.02**	**0.02**	**0.02**	**0.03**
orb.	NH-AHK$^-$	**0.01**	0.02	0.03	**0.04**	**0.12**	0.06	**0.02**	**0.04**	**0.09**	**0.16**
	DiGress	0.05	**0.00**	**0.02**	0.10	0.28	0.01	0.06	0.15	-	-
	GraphRNN	0.02					**0.00**				
	ER	0.04	0.08	0.11	0.13	0.29	0.10	0.20	0.28	0.46	0.70

Table 3: Evaluation of degree distributions, clustering distributions, and orbit distributions using the Maximum Mean Discrepancy (MMD) metric for each model and network.

B.6 Additional analysis

Here we report additional analysis on the *Community* dataset and samples of generated graphs. Given that the original *Community* networks consist of precisely two communities, each with an intra-community edge probability of 0.3 and inter-community edges with a probability of 0.0025, we partition the input graphs using the Clauset-Newman-Moore greedy modularity maximization algorithm[6]. Subsequently, we evaluate the intra-community and inter-community edge probabilities. The summarized average values are presented in Table 4. As we present the average values, we also include the values from the test set. Consequently, the optimal model is the one with a value closest to that of the test set. Discerning from the table, it becomes evident that exclusively the NH-AHK$^-$ model maintains its proximity to the test set, even as the scaling factor increases.

Scaling factor		1	1.5	2	4	8
intra-comm.	Test	0.30 (±0.01)	0.30 (±0.01)	0.30 (±0.01)	0.30 (±0.00)	0.30 (±0.00)
	NH-AHK$^-$	0.28 (±0.02)	0.28 (±0.02)	0.28 (±0.02)	0.28 (±0.02)	0.28 (±0.02)
	Digress	0.29 (±0.01)	0.27 (±0.13)	0.28 (±0.17)	-	-
	GraphRNN	0.27 (±0.08)	-	-	-	-
	ER	0.34 (±0.13)	0.29 (±0.13)	0.27 (±0.12)	0.26 (±0.17)	0.22 (±0.12)
inter-comm.	Test	0.003 (±0.00)	0.002 (±0.00)	0.002 (±0.00)	0.003 (±0.00)	0.002 (±0.00)
	NH-AHK$^-$	0.004 (±0.00)	0.004 (±0.00)	0.004 (±0.00)	0.004 (±0.00)	0.004 (±0.00)
	Digress	0.003 (±0.00)	0.014 (±0.02)	0.021 (±0.02)	-	
	GraphRNN	0.010 (±0.00)	-	-	-	-
	ER	0.083 (±0.01)	0.085 (±0.01)	0.085 (±0.00)	0.085 (±0.00)	0.084 (±0.00)

Table 4: Intra-community and inter-community edge probability on the *Community* dataset

Figure 7 depicts test and generated random samples. The top row presents two *Community* networks for each model, while the bottom row displays two *EGO* networks for each model.

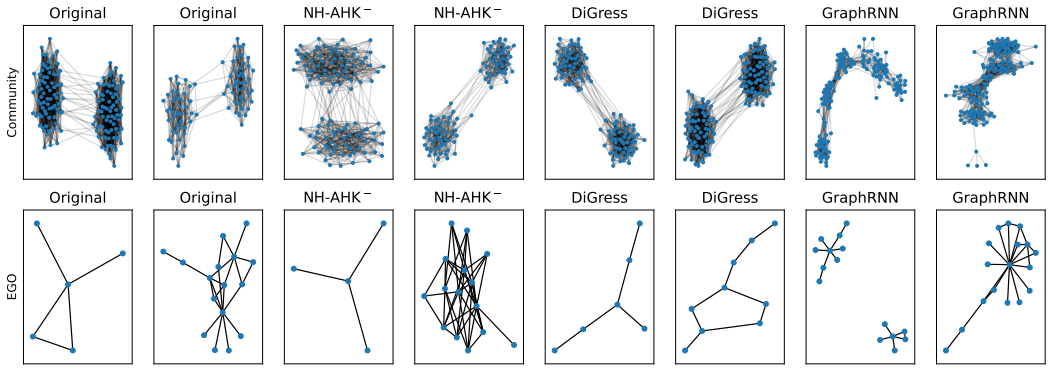


Figure 7: Samples from the test set, NH-AHK$^-$, *DiGress*, and *GraphRNN*, for *Community* and *EGO* networks

Figure 8 illustrates test and generated random samples as the scaling factor increases. It's worth noting that *GraphRNN* is not included due to its inability to generate larger graphs. While *DiGress* is not able to generate *Community* networks with scaling factors 4 and 8.

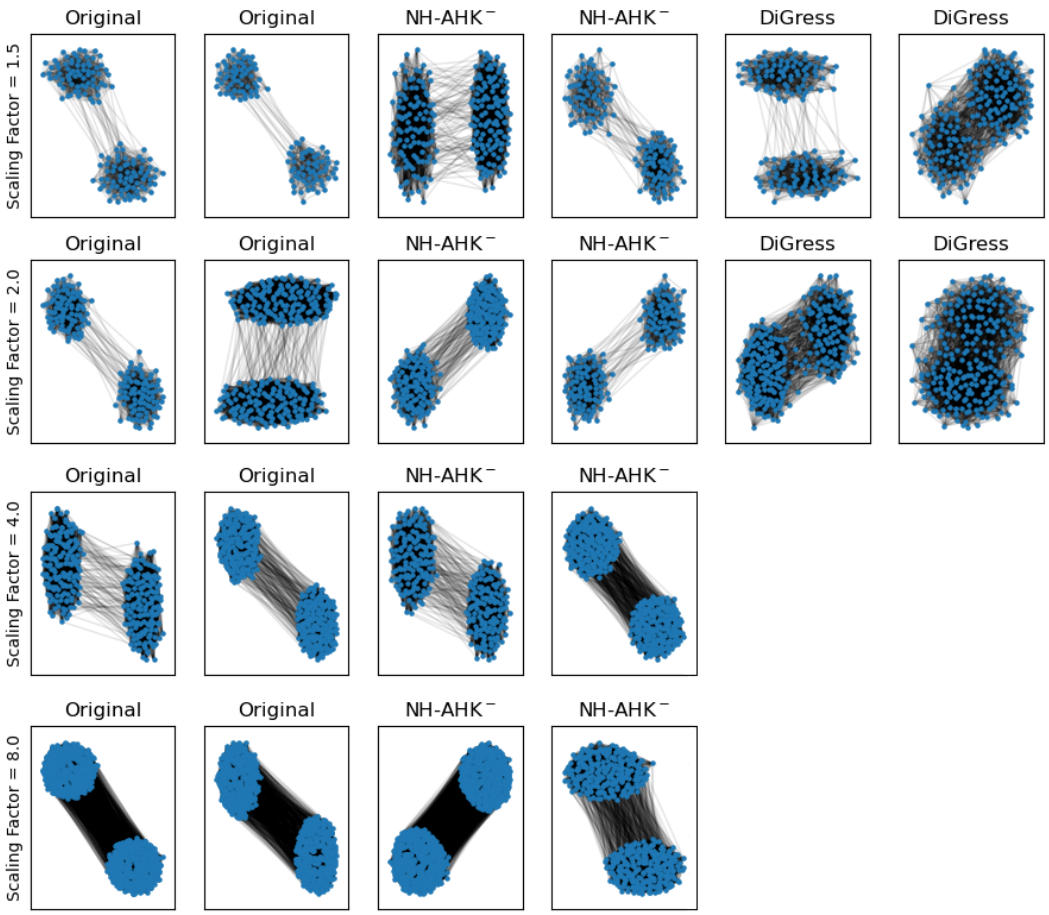


Figure 8: Samples from the test set, NH-AHK$^-$, *DiGress*, and *GraphRNN*, for *Community* as the scaling factor increases.

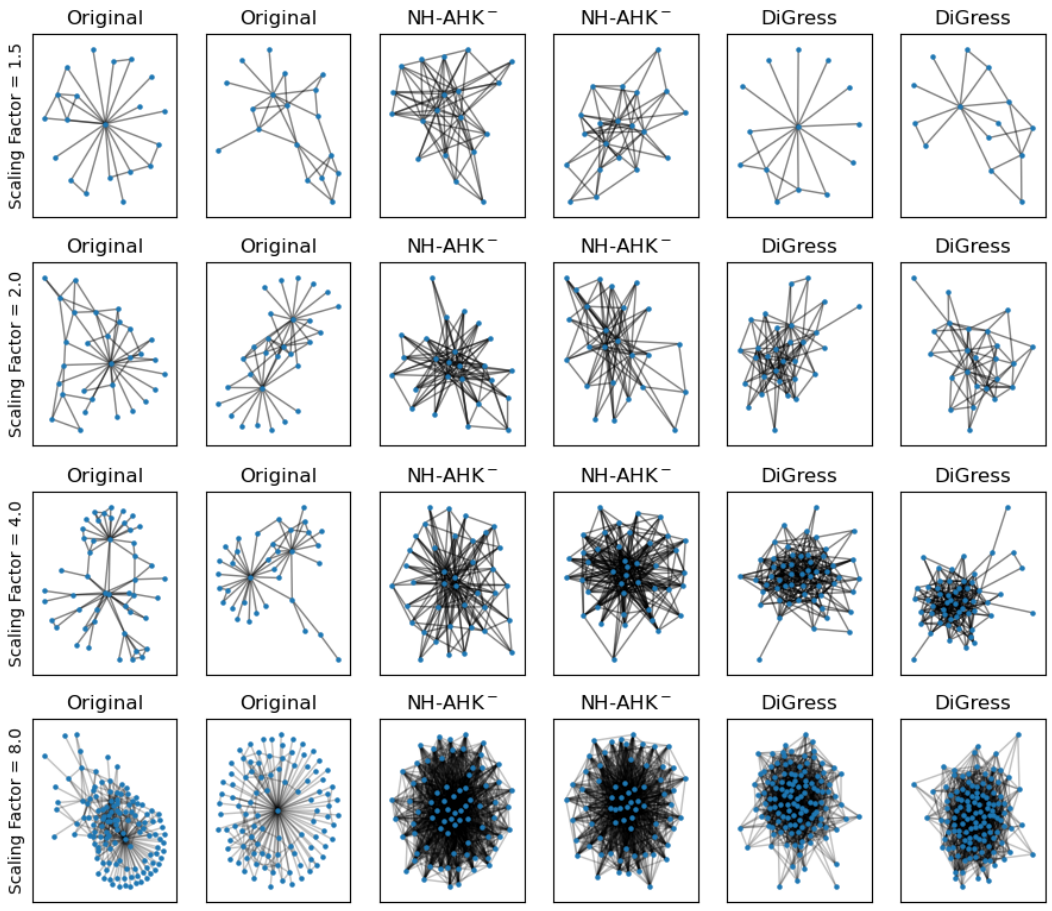


Figure 9: Samples from the test set, NH-AHK$^-$, *DiGress*, and *GraphRNN*, for *EGO* as the scaling factor increases.