ECAI 2024
U. Endriss et al. (Eds.)

doi:10.3233/FAIA240834

SaGess: A Sampling Graph Denoising Diffusion Model for Scalable Graph Generation

Stratis Limnios[a,*], **Praveen Selvaraj**[b], **Mihai Cucuringu**[c,a,d], **Carsten Maple**[b,a], **Gesine Reinert** [c,a] and **Andrew Elliott** [e,a]

[a]The Alan Turing Institute, London, UK
[b]University of Warwick, Coventry, UK
[c]Department of Statistics, University of Oxford
[d]Mathematical Institute, University of Oxford
[e]Department of Mathematics and Statistics, University of Glasgow, UK

Abstract. Denoising diffusion generative models are state-of-the-art methods for generating synthetic images that have also proved successful in tabular and graph synthetic data generation. However, their computational complexity has limited the application of these techniques to graph data, focusing usually on smaller graphs, such as those used in molecular modeling. In this paper, we propose SAGESS, a discrete denoising diffusion approach, which is able to generate large real-world networks. Through a generalized divide-and-conquer framework, SAGESS overcomes the scaling limitations of the diffusion model DIGRESS, by sampling a covering of subgraphs of the initial graph, training a DIGRESS module, and finally reconstructing a synthetic graph using the subgraphs that have been generated using the DIGRESS module. We evaluate the quality of the synthetic data sets against several competitor methods by comparing graph statistics between the original and synthetic samples, as well as evaluating the utility of the synthetic data set produced by using it to train a task-driven model, namely link prediction. In our experiments, SAGESS outperforms most of the one-shot state-of-the-art graph generating methods by a significant factor, both on the graph metrics and on the link prediction task.

1 Introduction

Synthetic data are key to many methods in machine learning and statistics, and their generation has sparked a significant amount of attention in recent years. Beyond generating appealing synthetic images using Dall-E or asking ChatGPT to write prose, many real world applications benefit greatly from synthetic data generation, for tasks including data augmentation in the training of classifiers/regressors [29], privacy protection of sensitive data [5] or removing bias from data sets [27]. Synthetic graph generation for modeling social interactions, generating new chemical compounds, or forecasting transactions requires efficient methods.

Many complex data sets can be represented as networks, and hence synthetic graph generators are of particular interest. Often these networks are viewed as realizations of a random process. The design of generative models for random graphs has a rich history, coming from probabilistic and structural assumptions with traditional methods such as early work in Erdős-Rényi graphs [10], stochastic block models [15], exponential random graphs [23] or the Barabási-Albert model [4]. These methods often oversimplify the underlying complex structure of the graphs and are then not able to capture the distributions arising from real-world scenarios. Thus, more recently there has been an increasing interest in designing deep models for synthetic graph generation, which allows for more flexible algorithms that are able to capture the intricacies of real networks with complex dependencies between the edges. Among approaches that have been proposed for graph generation sit autoregressive methods which generate nodes and edges step by step. Examples include GraphRNN [34] and GRAN [18], which improves modeling of long-term dependencies using a graph-based attention mechanism. Additional methods include autoencoder based approaches, such as GraphVAE [25], and adversarial approaches such as [6]. These are only a few methods; detailed reviews are found e.g. in [11] or in [12].

Formally, creating a synthetic graph generator is equivalent, either explicitly or implicitly, to estimating, or sampling from, a probability distribution over space of possibly directed, possibly weighted and perhaps attributed graphs. When many independent realizations from the unknown distribution are available, then standard estimation methods are often successful. Yet, many real world graph data sets consist of only a single graph, either because it is expensive to measure, or because of the nature of the data, for example there may only exist one instance of a global social network. Thus, there is interest in the more difficult task of approximating the underlying probability distribution only viewing a single sample. Classical models often navigate this task by making strong assumptions, such as an Erdős-Rényi model, which assumes that the edges are independent and identically distributed (i.i.d.). Recent deep learning methods often approach this challenge by the structured model of a GAE/GVAE or by learning the distribution over subsamples of nodes and edges; an example is NetGAN, an adversarial approach which relies on random walks [6]. Implicitly this approach assumes that the underlying process that generated the network is related to these samples.

From a different angle, after the superb performances of Denoising Diffusion Models [14, 26] on image generation, various papers have applied such models to graph learning, mostly for molecular generation such as GeoDiff [30] or chemical compound design [24]. More versatile models have been introduced, such as DIGRESS [28], a discrete denoising diffusion graph model that is able to generate high

* Corresponding Author. Email: slimnios@turing.ac.uk

quality graphs with node and edge attributes. One of the main assets of denoising diffusion models is not having to rely on adversarial training, but they still need a large data set to train. The implication of this is that models such as DiGress do not adapt well when the task is to generate one large graph from a single sample. HiGGS from [8] uses a hierarchical sampling approach to improve scalability; its adaptation using DiGress is called edge-DiGress. In a related fashion, EDGE [7] uses a discrete diffusion model in conjunction with a sparsification of the nodes by considering only a subset of the nodes as "active" at every step. [16] provide a hierarchical graph generation method based on explicit multinomial probabilities.

In this paper, instead we propose a denoising-based diffusion model by leveraging a divide-and-conquer scheme, which allows us to generate samples from larger single graphs than DiGress would permit.

Our main contributions can be summarized as follows.

• We introduce graph subsampling methods, to break large graphs into a trainable data set.

• We propose SaGess (SAmpling Graph dEnoiSing diffuSion model), a pipeline with a DiGress module which combines small samples to sample a large graph.

• We propose a task-driven evaluation by training link prediction GVAE on synthetic data and testing on real data.

The paper is structured as follows. We first set up the notations for the rest of the paper, as well as review the DiGress model, while also elaborating on its limitations. We then state the problem, and then present our solution framework SaGess, which unfolds in two sections: first, the sampling methods to obtain a training data set, and second, the reconstruction pipeline of the synthetic graph. Next, we present our experimental setup and results. Finally, we discuss potential limitations and future work. An ablation study as well as a discussion of the sampling methods, hyperparameters, and computational resources are found in the Supplementary Information (SI). [1] The code to our model is available at

https://github.com/Slimnios/SaGess.

2 Graph Diffusion Model and Scalability

Notations and definitions presented in this section will be used throughout this paper. Graphs in this paper are denoted as $G = (V, E)$ where V is the set of $|V| = n$ nodes and E is the set of $|E| = e$ edges. Each node is of one of a types, and each edge is of one of b types. We associate with G the matrices $X \in \mathbb{R}^{n\times a}$ where $X_{i,:}$ is the one-hot encoding for the type

of node i, and $E \in \mathbb{R}^{n\times n\times b}$ where $E_{i,j,:}$ is the one-hot encoding for the feature of the edge between nodes i and j. We denote by $P_k(V)$ the set of all k-point subsets of V and by $\mathcal{S}_k(G)$ the set of all possible subgraphs of G of size k. The subgraph of G induced by the nodes in S is denoted by $G[S]$; we also refer to such graphs as *node induced subgraphs*.

2.1 *Graph Generation using Discrete Denoising Diffusion Model:* DiGress

Here we review some key aspects of DiGress. DiGress [28] is currently one of the most efficient tools in graph generation. Taking as input a data set of a variety of graphs, it learns a denoising process in discrete space and is able to mimic the input graphs with remarkable precision. A key aspect of this method, and more generally graph generation methods based on diffusion models [13, 28], is the reliance on a discrete space noise scheduling. Indeed, instead of the standard Gaussian noising and denoising procedure, these frameworks propose to add discrete noise iteratively via random edge and node addition and deletion.

Traditionally, these models are based on a forward and a reverse Markov process. For a sample x, a forward process $q(x^{1:T}|x) = \prod_{1=t}^{T} q(x^t|x^{t-1})$ generates increasingly noisier samples up to getting white noise; here T is a hyperparameter. Then the model learns a reverse process $p_\theta(x^{1:T}) = p(x^T)\prod_{1=t}^{T} q(x^{t-1}|x^t)$ that aims to denoise x^T to obtain a synthetic sample.

The DiGress model takes as state space the set of node types and of edge types. DiGress then defines the transition probabilities from one state to another for nodes and edges through the noise matrices $[Q_X]^t_{i,j} = q(x^t = j|x^{t-1} = i)$ and $[Q_E]^t_{i,j} = q(e^t = j|e^{t-1} = i)$ where Q is chosen so that the Markov chain converges to the relative type frequencies in the initial population. Then, to get a noisy sample $G^t = (X^t, E^t)$ each node and each edge type is sampled from the categorical conditional distribution $q(G^t|G) = (X\bar{Q}^t_X, E\bar{Q}^t_E)$ with $\bar{Q}^t_X = Q^1_X \dots Q^t_X$ and $\bar{Q}^t_E = Q^1_E \dots Q^t_E$.

The denoising component of the DiGress model is a denoising neural network ϕ_θ parametrized by θ. It is trained by optimizing the cross-entropy loss (**c-e**) between the predicted probabilities $\hat{p} = (\hat{p}^X, \hat{p}^E)$ and the true graph G,

$$l(\hat{p}^G, G) = \sum_{1\le i\le n} \textbf{c-e}(x_i, \hat{p}^X_i) + \lambda \sum_{1\le i,j\le n} \textbf{c-e}(e_{ij}, \hat{p}^E_{ij}), \quad (1)$$

where $\lambda \in \mathbb{R}^+$ is a hyperparameter.

The trained network can be used to generate synthetic graphs via the estimation of the reverse diffusion $p_\theta(G^t|G)$ using $\hat{p}^G$, as follows. We put

$$p_\theta(x^{t-1}_i|x_i = x, G^t) = q(x^{t-1}_i|x_i = x, x^t_i)\mathbf{1}(q(x^t_i|x_i = x) > 0)$$

and

$$p_\theta(e^{t-1}_{ij}|e_{ij} = e, G^t) = q(e^{t-1}_{ij}|e_{ij} = e, e^t_{ij})\mathbf{1}(q(e^t_{ij}|e_{ij} = e) > 0),$$

and we set

$$p_\theta(x^{t-1}_i|G^t) = \sum_x p_\theta(x^{t-1}_i|x_i = x, G^t)\hat{p}^X_i(x),$$
$$p_\theta(e^{t-1}_{ij}|G^t) = \sum_x p_\theta(e^{t-1}_{ij}|e_{ij} = e, G^t)\hat{p}^E_{ij}(e),$$

to write

$$p_\theta(G^{t-1}|G^t) = \prod_{i=1}^{n} p_\theta(x^{t-1}_i|G^t) \prod_{1\le i,j\le n} p_\theta(e^{t-1}_{ij}|G^t).$$

The sampled G^{t-1} serves as input of the denoising network at the next time step. This derivation shows the complexity of the calculations; when the numbers of types, and the network, and T are all small, they are feasible, but large networks with many different types can pose a challenge. More details are in the Supplementary Information (SI).

Overall, DiGress can perform extremely well for generating small graphs with attributes. Unfortunately, the complexity of DiGress and the way the model is set up prevents it from being used it to sample one large graph. The complexity lies in the dimensions of the matrices $\mathbf{X}, \bar{\mathbf{Q}}^t_X, \mathbf{E}$ and $\bar{\mathbf{Q}}^t_E$ as at least the edge matrices scale in n^2 times the number of attributes. It is indeed specified in [28] that DiGress has complexity $\Theta(n^2)$ per layer, due to the attention scores and the predictions for each edge. Hence, not only we need

to store these matrices in memory for every diffusion step, but we also need to learn T dense matrices Q_E to define the Markov transitions. This is fairly reasonable for small graphs, but starts to become computationally difficult when the number of nodes in the graph increases. Moreover, supposing we are able to run DIGRESS on large graphs, we cannot train it on one single graph. DIGRESS, much like other deep-learning algorithms, thrives when having access to a large training set, which may not be available. Thus, there is considerable scope for amendments of the scalability issues of DIGRESS. This provides the motivation for proposing our framework based on sampling from large graphs.

3 A Sampling Graph Denoising Diffusion Model Generator (SAGESS)

This paper addresses the task to generate a single graph from a unique observation. We do not have a large training set of graphs $\mathcal{G}$ available to train any type of diffusion model. Our solution, which we denote as SAGESS and which is visualized in Figure 1, can be summarized as follows. First, we produce $\mathcal{G}$ – a set of graphs issued from the initial graph G. Next, we employ DIGRESS to create samples using the data set $\mathcal{G}$ as training data. Finally, we rebuild a graph from the trained diffusion model in a systematic fashion, paying particular care to match the node ids in the samples.

3.1 Graph Sampling and Covering

The first step is to construct a representative data set to train DIGRESS. We propose three sampling schemes to capture different structural properties of the graph. Each method addresses one or more properties of the graph, from local to more global ones. Depending on the real world graph to which we apply SAGESS, some sampling schemes might be more effective than others; for instance, ego networks might benefit more from a local sampling procedure, whereas communication data sets, for instance, might benefit more from more a global sampling procedure such as random walks, to capture long range dependence between edges. More discussion on the choice of sampling methods is found in Section 6.

Uniform Node Sampling (Unif): We sample a set of node induced subgraphs by selecting k sized subsets of nodes S_k uniformly at random. We note that this sampling scheme is invariant under permutation of the nodes. Based on results from [20], if p denotes the probability of selecting a node, then $\Theta(p^{-2}\log n \log(1/\delta))$ uniformly node induced subgraphs suffice for reconstruction with probability at least $1-\delta$. In our setting, $p = k/n$. To give a broad sense to the number of subgraphs we need to sample, for a graph with $n = 1,000$ nodes, suppose we select uniformly at random $2\log(n)$ nodes. We would then need $\Theta(4\log^3(n)\log(1/\delta))$ subgraphs of size 20 to reconstruct G. In practice we set the data set of samples to $10,000$ graphs. We provide an ablation study on this parameter as well as a graph coverage analysis in the SI.

Random Walk Node Induced Subgraph (RW): The second sampling method is based on sampling from subgraphs induced by random walks starting from every node in the graph G. The idea is straightforward: for each node v_1 of the n nodes in V we generate a k step random walk $w^k = \{v_1, \dots, v_k\}$ starting from v_1. Then we obtain the node induced subgraph from those walks $G[w_k]$. We repeat this construction d times. This gives us a data set $\mathcal{G} = \{G[w_1^k], \dots, G[w_{d\times n}^k]\}$, where d is a hyper-parameter controlling the number of sampled graphs per node. Using results by [3] and [9], heuristically, after order $n^2\log(n)$ steps of the walk, with high probability all nodes in the network are covered; see the SI for additional discussion.

2-hop Neighborhood Sampling (Ego): The last sampling method is based on sampling, from every node, a 2-hop neighborhood with random node deletion. As a 2-hop neighborhood can cover most of the graph, we aim for small samples to be able to train DIGRESS, with ideally more than one subgraph including each node. We fix the maximum size of a subgraph generated from the 2-hop neighborhood of a node to be k. If the neighborhood is larger than k, we delete (the integer part of) half of the nodes from the neighborhood uniformly at random. We repeat this until there are less than k nodes in the largest connected component; we denote this resulting node set by $\mathcal{N}^k(v)$. Again, we obtain its node induced subgraph $G[\mathcal{N}^k(v)]$. For k small enough, there is randomness in our modified 2-hop neighborhoods. We generate d subgraphs per node, with d defined previously, producing a data set $\mathcal{G} := \{G[\mathcal{N}_i^k(v_j)], i = 1, \dots, d, j = 1, \dots, n\}$. The algorithm is given in Algorithm 1. Similarly as for random walk sampling, heuristically, with high probability all nodes are covered after order $n^2\log n$ steps; see SI for details. [2] gives further theoretical guarantees for this sampling procedure.

Algorithm 1 Ego Sampling

Input: $G, d = \lfloor$ **number-of-subgraphs** $/n \rfloor$
Output: $\mathcal{G}$

for $i \in \{1, \dots, n\}$ **do**
  for $j \in \{1, \dots, d\}$ **do**
    $G_i = \mathcal{N}(v_i)$
    while $|V_i| > k$ **do**
      choose uniformly $\lfloor \frac{|V_i|}{2} \rfloor$ nodes from V_i; call this set S_i
      delete the nodes in S_i from G_i
      retain the largest connected component of G_i
    end while
  end for
end for

Each of these subgraph sampling methods provides a graph training set for DIGRESS which is obtained from one single graph. Moreover, following [20], we produce enough samples to have a good chance to cover each node and edge in the uniform sampling scheme, with the heuristic that this should also be adequate for the denser sampling schemes. We also shuffle the training data set $\mathcal{G}$ before feeding it to DIGRESS. Next, we detail how SAGESS reconstructs the initial graph from the trained model.

3.2 Node Labeling and Reconstruction

Our SAGESS model first samples subgraphs to train DIGRESS, because DIGRESS is only able to generate small graphs from the training set $\mathcal{G}$ it learns from. As the nodes in those graphs, if not assigned attributes, are just rows in an adjacency matrix, without intervention in general it may not be possible to merge the resulting collection of small graphs into a single graph with uniquely identifiable nodes.

This is where the ability of DIGRESS to handle node attributes is important. We set the initial node ids as node features on the subgraphs to enable identification after generating synthetic samples. In detail, to every graph $G_i^k \in \mathcal{G}$ we associate a feature matrix $\mathbf{X_i} \in \mathbb{R}^{k\times n}$, where k is the number of nodes in G_i^k and n the number of nodes in G, so that $\mathbf{X_i}$ is a one-hot encoding of the node id

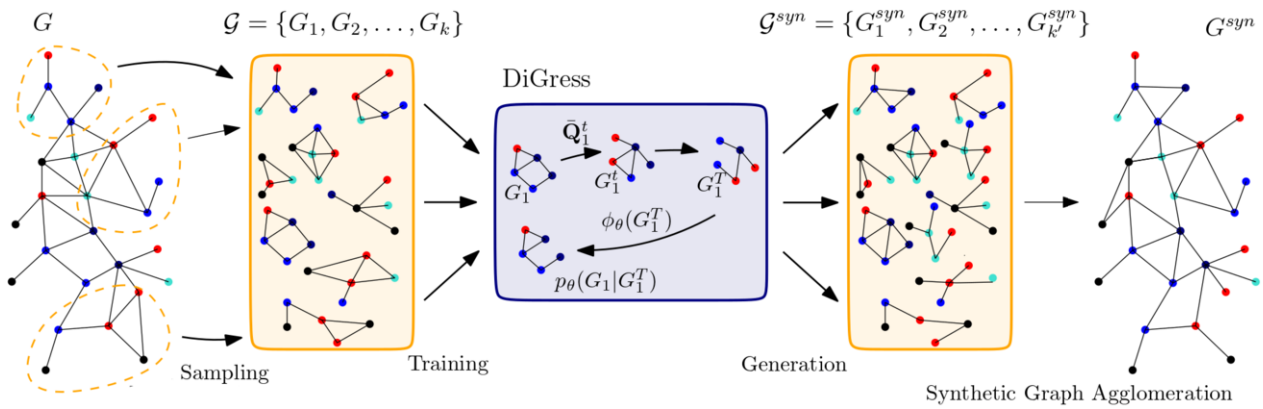

Figure 1: SAGESS Diagram

Algorithm 2 SAGESS

Input: $\mathcal{G}$**: Sample** $n \times d$ **samples using Unif/Ego/RW**
Output: G_{syn}

1: Train DIGRESS with $\mathcal{G}$
2: **while** $|E_{syn}| < |E|$ **do**
3: generate synthetic subgraph S_{syn}
4: add new unique edges from (S_{syn}) to G_{syn}
5: **end while**

in the initial graph. This enables SAGESS to learn the local graph structure as we identify the nodes with the attribute. We then generate enough small graph samples with DIGRESS and agglomerate the new edges and nodes uniquely until we match the edge count in the initial graph G. The agglomeration step unfolds as follows. We start with the first generated graph. Then as long as the number of edges, $|E|$, in the original graph is not reached, we generate a new synthetic graph G_{syn}^k with DIGRESS and take the graph union with the current graph. The sampling process stop when $|E_{syn}| > |E|$; if G_{syn}^l is the last graph sampled, we add all the new edges to G_{syn} regardless of whether the new edge count exceeds $|E_{syn}|$. As the generated subgraphs tend to be small, the overshoot tends to be small also. The SAGESS algorithm is shown in Algorithm 2.

Combining these stages gives the SAGESS pipeline a diffusion model which can generate large graphs. Further, as it builds on DIGRESS, SAGESS comes with theoretical guarantees inherited from DIGRESS, in particular the permutation equivariant architecture and a permutation invariant loss. Exchangeability of the generated distributions then follows under the exchangeable SAGESS sampling schemes, permuting the order of the samples in the input to destroy any potential sequential dependence. These properties ensure that SAGESS can learn efficiently from the data.

4 Experimental Evaluation

Here we evaluate our framework against state-of-the-art graph generation methods on four real world data sets and one synthetic data set. First, we compare graph statistics to evaluate the quality of the generated graphs. Then we train a Variational Graph Auto-Encoder (GVAE) to evaluate the utility of the synthetic data generated on a link prediction task.

4.1 Benchmark models

To benchmark our approach, we compare against several established competitor methods which construct synthetic data based on a single sample. To make a meaningful comparison, we have selected methods from multiple different generation approaches, from classical approaches, through adversarial approaches, and approaches based on a low-rank approximation. They are as follows:

DCSBM [17] A classic approach from the network science literature, the so-called degree corrected stochastic block model[1] assumes a classic stochastic block model, where the nodes are divided into blocks, and the probability of a connection between a pair of nodes is a function of their block membership and their degree.

NetGAN [6] An adversarial approach leveraging the GAN framework, to learn the distribution of biased random walks which are then combined via a transition matrix into a sampled graph. Edges in the sampled graph are sampled (mostly) uniformly at random. [2]

EDGE [7] Another attempt at scalability for graph diffusion model generator is EDGE, which initially generates a node degree distribution and the induced adjacency matrix, preserving sparsity. Then it applies a diffusion model to this setup while keeping the sparcity during the noising and denoising process. [3]

CELL [22] A modification of NetGAN which replaces the GAN formulation with a formulation based on a low-rank approximation, see paper for full discussion. In this formulation, sampled graph edges are again sampled in an edge independent manner.[4]

4.2 Data sets

We evaluate our method on four real world data sets from the torch geometric package[5] and one synthetic data set, including sparse/less sparse and directed/undirected graphs:

EuCore: An e-mail communication network of a large European research institution, from [33]. Nodes indicate members of the institution, and an edge between a pair of members indicates that they exchanged at least one email. This graph has $1,005$ nodes and $16,706$ edges, with an edge density of 0.03311.

Cora: A citation data set from [32]. Every node is an article and an edge links two nodes if one cites the other. It consists of a directed graph with $2,708$ nodes and $10,556$ edges, with an edge density of 0.00288.

Wiki: A data set from Wikipedia pages from [31] with $2,405$ nodes and $12,761$ undirected edges, with an edge density of 0.004414.

Facebook: This data set consists of friend lists from Facebook published in [19]. This data set initially contains 10 graphs, but we

[1] Implementation: https://github.com/microsoft/graspologic
[2] Implementation: https://github.com/danielzuegner/netgan
[3] Implementation:https://github.com/tufts-ml/graph-generation-edge
[4] Implementation: https://github.com/hheidrich/CELL
[5] https://pytorch-geometric.readthedocs.io/en/latest/modules/datasets.html

only use the largest one (second graph) which has $1,045$ nodes and $27,755$ undirected edges, with an edge density of 0.05088.
SBM: This is a standard directed Stochastic Block Model generated graph, with 4 blocks of sizes 400. We set the inner-cluster density to 0.2 and the across-cluster density to 0.01. It has edge density of 0.05731.

4.3 Experiments

We evaluate our framework using three different experimental settings. In our first experiment, we evaluate how close the structural properties of the generated graphs from each of the methods are to the original graph. For evaluation we choose the set of standard metrics from [22] such as number of nodes (ignoring isolated nodes), average local clustering coefficient, assortativity and triangle count. As some generation methods fix the number of nodes and others fix the number of edges, we report these numbers also but do not assess the methods on them.

To further explore our pipeline, we compare the effectiveness of each of our sampling schemes using the same approach. We produce and compare synthetic graphs for all data sets for the random walk (RW), and 2-hop modified neighborhood (Ego) sampling schemes. For the uniform (Unif) sampling scheme, we compare only on EuCore, Facebook and SBM, as this sampling scheme struggles on the remaining data sets, due to their sparsity.

As the induced subgraphs in the training set from the uniform sampling tend to have very few edges (of order of magnitude n), SAGESS may struggle to find the right amount of edges to match the edge count of the initial graph. This observation illustrates the benefit of selecting an appropriate sampling scheme.

The second experiment consists of a utility test; we want to evaluate the usefulness of the synthetic data set produced. Hence, we propose to train a Variational Graph Auto-Encoder for link prediction; the GVAE is trained on the generated synthetic data.

Once trained, we obtain a latent space from the synthetic data set on the set of nodes belonging to the real data set. The evaluation is the following: we compute the probability of each possible edge using the GVAE encodings and evaluate them on 90% of the edges of the real graph. In order to evaluate the precision and obtain the precision scores in Table 2 we sample as many negative edges at random as there are positive ones in the training data set. This can also be interpreted as a test on how efficient the synthetic data set can be, when employed for a data augmentation task.

Finally, we also explore the ability of SAGESS to produce meaningful smaller graph samples. Indeed, it might not be needed to generate a graph of the same size as the initial graph G. Our goal is to demonstrate that SAGESS can also generate smaller scale synthetic graphs in terms of edges that are structurally similar to the initial graph. We generate graphs of sizes ranging from 10% to 100% of the number of edges in the initial graph, and observe the computed metrics; Figure 2 shows that they approach the true values. All experimental parameters and extended analysis can be found in the SI.

4.4 Runtimes, Comments and Improvements

The implementation of SAGESS has been optimised to mitigate two main potential bottlenecks. The first one is related to the feature matrix being one-hot encoded; this results in a sparse matrix that could only be addressed by a single core. This bottleneck has been addressed, so that SAGESS is able to assign multiple workers to the dataloader. The second bottleneck arises from the subgraph sampling procedure. As subgraphs are sampled one node at a time, although uniform and random sampling are very fast, ego-sampling can be quite time-intensive, depending on the data set. This bottleneck has been navigated by a modification which allows to sample subgraphs for multiple nodes in parallel.

5 Results

Throughout our experiments, we have shown that SAGESS is a powerful tool which can create high quality synthetic graphs. This is demonstrated in our first experiment, where the statistics on the synthetic graphs (Table 1) produced by SAGESS perform well in comparison to the benchmark models, across all the benchmark data sets. In particular, the SAGESS-RW method performs well in the first four data sets, although it struggles to some extent in the synthetic SBM data set. It is worth noticing that SAGESS tends to generate local structures, such as triangles, which are closer to the input network than those of the other baselines. This is due to DIGRESS being efficient at sampling motifs, and the subgraphs used for training contain many of them due to the nature of the sampling, which samples subgraphs with interdependencies between the edges. It is important to point out that even if we train on local substructures via the subgraphs, we manage to accurately obtain global metrics like characteristic path length or clustering coefficient. In contrast, EDGE, by design, will match degree related statistics accurately but will perform poorly on the global and structural metrics.

In our second experiment, link prediction, with the results presented in Table 2, we find that SAGESS overall performs consistently compared to the other methods. We also notice that while SAGESS is close to other methods in terms of statistics on some data sets, the utility performance on the link prediction is somewhat more diverse. For example, while the DCSBM method performs well in terms of graph statistics on the synthetic SBM data set (see Table 1), SAGESS still outperforms it by a significant factor in the link prediction task. As an aside, Table 2 in the SI shows the proportion of edges which are present in the real data as well as in the synthetic data; absent edges are not included in this statistic. It shows that SAGESS generates samples with considerable amount of overlap with the underlying network. Whether or not this feature is desirable depends on the application task. Table 3 in the SI gives the densities of the networks; Facebook and SBM are of an order of magnitude denser than EuCore, Cora and Wiki. We do not observe any strong correlation between density and performance.

Figure 2 indicates that SAGESS can generate graphs with a substantially smaller number of edges, while maintaining reasonable approximations of the graph statistics. For example, smaller samples have connectivity and structural properties that are similar to the initial graph even with a number of edges which is only 40% the number of edges in the initial network. This is not only an interesting feature of the SAGESS sampling scheme/generator, it is also useful from a computational perspective, when one needs to deploy it on machines with limited resources or shorter time frames, especially as the complexity of graph operations often depends on the number of edges as well as nodes.

In Table 3 we compare the runtimes between our framework (using Ego sampling) with the improvements mentioned in the previous section. The runtimes reported consist of the sum of both training of the model and generation of a synthetic graph. For our framework the sampling step is also included in the runtimes. We note that the runtimes are affected mainly by the number of nodes in the graph.

Table 1: Comparing network statistics of graphs generated from each synthetic data method on five different graph data sets. This table includes the graph statistics of Table 2 in [22], from a similar experiment, with cluster coef. denoting the average local clustering coefficient and CPL denoting the characteristic path length; we also report the number of nodes and the number of edges. The text in bold **ABC** indicates the closest statistic to the real data, whereas the text underlined $\underline{ABC}$ indicates the second closest.

Method	num nodes	num edges	num triangles	num squares	max deg	cluster coef.	assort.	power law exp	CPL
EuCore (Real)	1005	16,706	105,461	4,939,311	346	0.39935	−0.01099	1.3621	2.58693
SAGESS-Uni	939	16,716	114,900	6,280,664	287	0.34024	−0.06321	1.35697	2.49139
SAGESS-RW	878	16,709	131,429	6,995,335	**351**	0.42338	−0.04234	1.35192	2.50512
SAGESS-Ego	867	16,707	**114,593**	**5,257,738**	342	**0.39000**	−0.023969	1.3296	2.43646
NetGAN	986	16,064	62,278	2,505,330	279	0.25569	−0.06196	1.34179	2.48189
CELL	1005	16,064	74,251	3,336,294	273	0.29808	−0.07655	**1.36279**	**2.56782**
EDGE	1003	15,915	47,987	2,549,947	217	0.130935	−0.239587	1.359872	2.503124
DCSBM	951	15,906	75,743	3,699,308	305	0.19673	**−0.010515**	1.35087	2.47134
Cora (Real)	2708	10,556	1,630	4,664	168	0.24067	−0.06587	1.93230	6.31031
SAGESS-RW	2548	10,557	1,806	14,952	220	**0.25728**	−0.05969	1.84290	5.43069
SAGESS-Ego	2540	10,562	**1,696**	7,923	171	**0.23911**	**−0.06868**	**1.92240**	5.77869
NetGAN	2485	10,138	932	2,394	128	0.15687	−0.07384	1.86148	5.86039
CELL	2708	10,556	521	1,295	97	0.07930	−0.08226	1.88642	**6.03067**
EDGE	2708	6,053	83	687	**168**	0.00422	−0.05556	1.80236	4.82168
DCSBM	2621	10,097	2,380	**2,809**	237	0.06991	−0.01709	1.85581	4.47947
Wiki (Real)	2405	12,761	23,817	407,302	263	0.37581	−0.07875	1.54227	3.65161
SAGESS-RW	2348	12,763	**26,296**	**466,120**	280	0.38389	−0.09282	**1.54076**	3.53405
SAGESS-Ego	2275	12,763	26,897	506,620	312	**0.37530**	−0.10494	1.55261	3.47834
NetGAN	2405	11,596	10,701	104,243	241	0.20179	−0.10076	1.52944	3.60293
CELL	2357	11,592	10,136	121,014	258	0.22965	−0.10769	1.54027	**3.62619**
EDGE	2405	12,701	4,915	85,516	265	0.041027	**−0.078539**	1.5151	3.327667
DCSBM	2251	11,595	9,439	167,655	**263**	0.08492	−0.01821	1.54596	3.45205
Facebook (Real)	1045	27,755	446,846	34,098,662	1044	0.57579	−0.02543	1.28698	1.94911
SAGESS-Uni	1043	27,758	**429,428**	**35,261,545**	999	0.52098	−0.01607	1.29003	2.00800
SAGESS-RW	1009	27,764	490,844	43,006,252	1001	**0.56138**	**−0.02266**	1.29398	**1.96014**
SAGESS-Ego	1005	27,761	515,928	45,421,130	295	0.43074	0.34074	1.29381	2.65926
NetGAN	1045	27,755	262,574	15,635,262	849	0.39773	−0.01821	1.27429	2.13730
CELL	1045	27,755	250,968	14,855,676	474	0.30854	0.12788	1.27490	2.38650
EDGE	1045	27,253	155,502	9,551,169	**1010**	0.16818	−0.084674	1.28118	1.97056
DCSBM	1041	27,092	339,448	26,714,948	733	0.37549	0.07125	**1.28845**	2.33021
SBM (Real)	1600	73,312	344,574	20,955,308	155	0.15418	−0.00188	3.58894	2.08276
SAGESS-Uni	1600	73,313	**342,639**	21,054,221	**164**	0.14830	0.03572	2.16732	2.04884
SAGESS-RW	1600	73,367	490,663	34,384,218	207	0.20062	0.22337	1.70495	2.13963
SAGESS-Ego	1600	73,326	366,162	22,979,792	174	**0.15622**	0.07177	1.89330	2.06118
NetGAN	1600	73,312	367,143	22,775,320	144	0.16054	0.19039	2.39375	2.12747
CELL	1600	73,312	341,351	**20,783,575**	139	0.15186	**−0.00156**	2.72805	2.08060
EDGE	1600	69,305	122,382	8,285,350	138	0.05873	−0.00185	1.74766	1.96714
DCSBM	1600	73,357	353,934	21,794,585	130	0.15671	−0.00343	**2.79204**	**2.08274**

This is an expected outcome as increasing the number of nodes increases the one-hot encoding representation size. We observe though that SAGESS outperforms the other methods by a significant factor when it comes to the denser graphs. Hence, in scenarios with very dense graphs SAGESS is fast and, in our comparison, the best option.

SAGESS comes with a user-friendly code repository containing scripts for environment setup, training and dataset customization. To ensure compatibility with HPC clusters and long-term usability, it also includes scripts for setting up a Docker environment and for submitting training jobs using Slurm workload management. This allows SAGESS to be run seamlessly on different systems and maintain long-term compatibility.

We present and discuss more results in the SI, including an ablation study of , and a graph coverage analysis of the sampling methods.

6 Discussion and Conclusion

In this paper, we propose SAGESS, a sampling-based denoising diffusion probabilistic model, based on DIGRESS. It is presented as a framework that is able to build a training data set from a single graph observation, and use it to generate a synthetic graph from a collection of generated synthetic subgraphs. Three options for subgraph sampling are available. The uniform sampling option is theoretically the fastest for achieving coverage, but it does not work well for very sparse networks. The random walk sampling option is relatively fast and achieves good results in particular for link prediction. The 2-hop neighborhood sampling method often matches the network statistics of the underlying graph best, but it is computationally intensive. Thus, the choice of sampling method depends on density, task, and computational budget.

We have shown throughout a variety of experiments that SAGESS outperforms established methods on several real world data sets, not only by considering standard graph statistics, but also by showcasing the adaptability and utility of our framework. In future work we shall

Table 2: Link prediction using GVAE on the benchmark data sets, trained on the synthetic, tested on real.

Method	Score	EuCore	Cora	Wiki	FB	SBM
SAGESS	AUC	0.846	-	-	0.860	0.680
Unif	AP	0.833	-	-	0.842	0.644
SAGESS	AUC	0.846	**0.895**	**0.884**	**0.899**	0.692
RW	AP	0.834	**0.895**	**0.894**	**0.886**	0.631
SAGESS	AUC	0.846	0.888	0.871	0.889	0.699
Ego	AP	0.833	0.890	0.886	0.877	0.645
NetGAN	AUC	0.812	0.510	0.561	0.870	0.698
	AP	0.816	0.510	0.583	0.850	0.641
EDGE	AUC	0.750	0.506	0.525	0.500	0.515
	AP	0.774	0.510	0.536	0.495	0.519
CELL	AUC	**0.857**	0.790	0.841	0.871	**0.700**
	AP	**0.846**	0.806	0.864	0.847	**0.648**
DCSBM	AUC	0.656	0.510	0.497	0.498	0.538
	AP	0.638	0.504	0.511	0.498	0.526

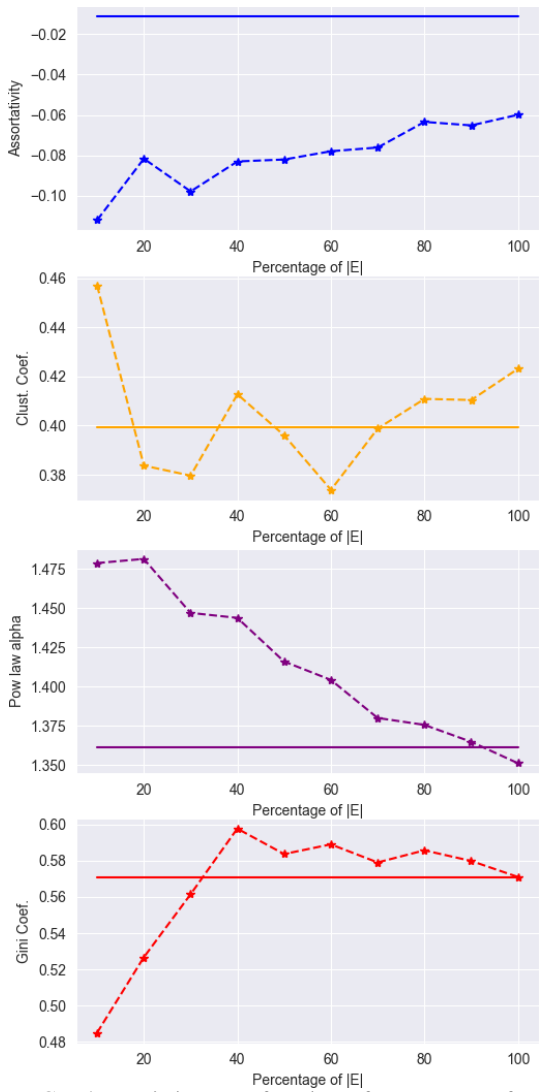

Figure 2: Graphs statistics as a function of percentage of sampled edges from the EuCore data set using the SAGESS-RW model. The y-axis is scaled to data; the amount of variation of the statistics is relatively small.

also assess SAGESS against the concurrent DIGRESS-based alternative: HIGGS [8]. In contrast to HIGGS, SAGESS does not assume an underlying community structure, and in contrast to EDGE [7],

Table 3: Methods runtimes for training and generation.

Method	EuCore	Cora	Wiki	Facebook	SBM
SaGess-Ego	$1h20$	$7h11$	$6h04$	$1h23$	$2h04$
NetGAN	$5h24$	$1h42$	$2h09$	$10h47$	$6h24$
EDGE	$1d15h$	$1d13h$	$5h16$	$1d18h$	$2d15h$
CELL	$5h19$	$35m$	$40m$	$8m$	$8h47$

SAGESS does not require an explicit model for the node degrees. However, a limitation of SAGESS is that it still does not scale well with the number of nodes in the initial graph in terms of memory. This is due to the one-hot-encoding of the graph node ids passed in DIGRESS. On the same note, as the number of nodes increases, the number of subgraph samples needed to train SAGESS also increases, adding to the time complexity of DIGRESS. In essence the memory complexity depends on the size of a subsample, resulting in $O(k^2 + kn)$ where k is the size of the subgraph. Additionally, while this method could trivially deal with edge attributes as DIGRESS does, it cannot handle node attributes since the attribute option is already used by the node ids to represent the nodes of the real graph.

SAGESS is a pure graph generator; the only node "feature" used is a one-hot encoding of the node ids. This feature is required to assemble the synthetic network from the subgraphs. Of course nodes can have features, though, and these would be attached directly to the node ids (enlarging the 1-hot encoding) to avoid feature conflicts [such as a bird on 4 legs], but this will increase computations. In practice instead a 2-stage procedure is envisaged, first generating synthetic networks and then adding synthetic node features.

Although DiGress allows for the use of classifier-based guidance or classifier-free guidance, see [21], the one-hot encoding makes these options impractical for node features. Classifier-free guidance could steer SAGESS to generate more diverse samples, but at non-negligible computational cost.

An advantage of SAGESS is that it can be extended to address other related problems. While SAGESS has been designed to generate graphs from a single sample, it can be extended to data sets with multiple graphs in the case where the nodes are identifiable, by simply designing a sampling scheme which first samples the graph and then samples the subgraph using an appropriate scheme, although we leave this avenue for future work. It will also be left as future work to investigate possible applications on signed networks or even time dependent edges, as one can add time as an edge attribute. This could lead to a more complex generation scheme that could, for instance, include an auto-regressive module to evaluate time dependence on the newly encountered edges. We would also be eager to extend our framework to other graph generation methods that handle node attributes to generalize the ability of models to train on a single observation.

Finally, some caution is advised. Inference based on synthetic data particularly in sensitive applications such as health or risk analysis should use a number of different synthetic data generators before reaching a conclusion. Moreover, vigilance is advised to detect malicious applications of synthetic data generation, such as passing off fake data as real.

Acknowledgements

This work was supported in part by JADE: Joint Academic Data science Endeavour - 2 under the EPSRC Grant EP/T022205/1, & The Alan Turing Institute under EPSRC grant EP/N510129/1.

References

[1] *Supplementary Information: SaGess: A Sampling Graph Denoising Diffusion Model for Scalable Graph Generation*, Aug. 2024. Zenodo. doi: 10.5281/zenodo.13350580. URL https://doi.org/10.5281/zenodo.13350580.

[2] W. Ali, A. E. Wegner, R. E. Gaunt, C. M. Deane, and G. Reinert. Comparison of large networks with sub-sampling strategies. *Scientific reports*, 6(1):28955, 2016.

[3] N. Alon, C. Avin, M. Koucky, G. Kozma, Z. Lotker, and M. R. Tuttle. Many random walks are faster than one. In *Proceedings of the Twentieth Annual Symposium on Parallelism in Algorithms and Architectures*, pages 119–128, 2008.

[4] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[5] B. K. Beaulieu-Jones, Z. S. Wu, C. Williams, R. Lee, S. P. Bhavnani, J. B. Byrd, and C. S. Greene. Privacy-preserving generative deep neural networks support clinical data sharing. *Circulation: Cardiovascular Quality and Outcomes*, 12(7):e005122, 2019.

[6] A. Bojchevski, O. Shchur, D. Zügner, and S. Günnemann. Netgan: Generating graphs via random walks. In *International Conference on Machine Learning*, pages 610–619. PMLR, 2018.

[7] X. Chen, J. He, X. Han, and L.-P. Liu. Efficient and degree-guided graph generation via discrete diffusion modeling. *arXiv preprint arXiv:2305.04111*, 2023.

[8] A. O. Davies, N. S. Ajmeri, et al. Hierarchical gnns for large graph generation. *arXiv preprint arXiv:2306.11412*, 2023.

[9] R. Elsässer and T. Sauerwald. Tight bounds for the cover time of multiple random walks. *Theoretical Computer Science*, 412(24):2623–2641, 2011.

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

[11] F. Faez, Y. Ommi, M. S. Baghshah, and H. R. Rabiee. Deep graph generators: A survey. *IEEE Access*, 9:106675–106702, 2021.

[12] X. Guo and L. Zhao. A systematic survey on deep generative models for graph generation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[13] K. K. Haefeli, K. Martinkus, N. Perraudin, and R. Wattenhofer. Diffusion models for graphs benefit from discrete state spaces. *arXiv preprint arXiv:2210.01549*, 2022.

[14] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[15] P. W. Holland, K. B. Laskey, and S. Leinhardt. Stochastic blockmodels: First steps. *Social Networks*, 5(2):109–137, 1983. ISSN 0378-8733. doi: https://doi.org/10.1016/0378-8733(83)90021-7. URL https://www.sciencedirect.com/science/article/pii/0378873383900217.

[16] M. Karami and J. Luo. Higen: Hierarchical multi-resolution graph generative networks. *arXiv preprint arXiv:2303.03293*, 2023.

[17] B. Karrer and M. E. Newman. Stochastic blockmodels and community structure in networks. *Physical review E*, 83(1):016107, 2011.

[18] 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. *Advances in Neural Information Processing Systems*, 32, 2019.

[19] J. McAuley and J. Leskovec. Learning to discover social circles in ego networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 539–547, Red Hook, NY, USA, 2012. Curran Associates Inc.

[20] A. McGregor and R. Sengupta. Graph Reconstruction from Random Subgraphs. In M. Bojańczyk, E. Merelli, and D. P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 96:1–96:18, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-235-8. doi: 10.4230/LIPIcs.ICALP.2022.96. URL https://drops.dagstuhl.de/opus/volltexte/2022/16437.

[21] M. Ninniri, M. Podda, and D. Bacciu. Classifier-free graph diffusion for molecular property targeting. *arXiv preprint arXiv:2312.17397*, 2023.

[22] L. Rendsburg, H. Heidrich, and U. V. Luxburg. NetGAN without GAN: From random walks to low-rank approximations. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 8073–8082. PMLR, 13–18 Jul 2020.

[23] G. Robins, P. Pattison, Y. Kalish, and D. Lusher. An introduction to exponential random graph (p*) models for social networks. *Social Networks*, 29(2):173–191, 2007. ISSN 0378-8733. doi: https://doi.org/10.1016/j.socnet.2006.08.002. URL https://www.sciencedirect.com/science/article/pii/S0378873306000372. Special Section: Advances in Exponential Random Graph (p*) Models.

[24] A. Schneuing, Y. Du, C. Harris, A. Jamasb, I. Igashov, W. Du, T. Blundell, P. Lió, C. Gomes, M. Welling, et al. Structure-based drug design with equivariant diffusion models. *arXiv preprint arXiv:2210.13695*, 2022.

[25] M. Simonovsky and N. Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part I 27*, pages 412–422. Springer, 2018.

[26] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015.

[27] B. van Breugel, T. Kyono, J. Berrevoets, and M. van der Schaar. Decaf: Generating fair synthetic data using causally-aware generative networks. *Advances in Neural Information Processing Systems*, 34:22221–22233, 2021.

[28] C. Vignac, I. Krawczuk, A. Siraudin, B. Wang, V. Cevher, and P. Frossard. Digress: Discrete denoising diffusion for graph generation. In *The Eleventh International Conference on Learning Representations*, 2022.

[29] S. C. Wong, A. Gatt, V. Stamatescu, and M. D. McDonnell. Understanding data augmentation for classification: when to warp? In *2016 international conference on digital image computing: techniques and applications (DICTA)*, pages 1–6. IEEE, 2016.

[30] M. Xu, L. Yu, Y. Song, C. Shi, S. Ermon, and J. Tang. Geodiff: A geometric diffusion model for molecular conformation generation. In *International Conference on Learning Representations*, 2021.

[31] R. Yang, J. Shi, X. Xiao, Y. Yang, J. Liu, and S. S. Bhowmick. Scaling attributed network embedding to massive graphs. *arXiv preprint arXiv:2009.00826*, 2020.

[32] Z. Yang, W. Cohen, and R. Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International Conference on Machine Learning*, pages 40–48. PMLR, 2016.

[33] H. Yin, A. R. Benson, J. Leskovec, and D. F. Gleich. Local higher-order graph clustering. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, page 555–564, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450348874. doi: 10.1145/3097983.3098069. URL https://doi.org/10.1145/3097983.3098069.

[34] J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec. GraphRNN: Generating realistic graphs with deep auto-regressive models. In *International Conference on Machine Learning*, pages 5708–5717. PMLR, 2018.