NetGAN: Generating Graphs via Random Walks

Aleksandar Bojchevski [*1] **Oleksandr Shchur** [*1] **Daniel Zügner** [*1] **Stephan Günnemann** [1]

Abstract

We propose NetGAN – the first implicit generative model for graphs able to mimic real-world networks. We pose the problem of graph generation as learning the distribution of biased random walks over the input graph. The proposed model is based on a stochastic neural network that generates discrete output samples and is trained using the Wasserstein GAN objective. NetGAN is able to produce graphs that exhibit well-known network patterns without explicitly specifying them in the model definition. At the same time, our model exhibits strong generalization properties, as highlighted by its competitive link prediction performance, despite not being trained specifically for this task. Being the first approach to combine both of these desirable properties, NetGAN opens exciting avenues for further research.

1. Introduction

Generative models for graphs have a longstanding history, with applications including data augmentation, anomaly detection and recommendation (Chakrabarti & Faloutsos, 2006). Explicit probabilistic models such as Barabási-Albert or stochastic blockmodels are the de-facto standard in this field (Goldenberg et al., 2010). However, it has also been shown on multiple occasions that our intuitions about structure and behavior of graphs may be misleading. For instance, heavy-tailed degree distributions in real graphs were in strong disagreement with the models existing at the time of their discovery (Barabási & Albert, 1999). More recent works like Dong et al. (2017) and Broido & Clauset (2018) keep bringing up other surprising characteristics of real-world networks that question the validity of the established models. This leads us to the question: "How do we define a model that captures all the essential (potentially still unknown) properties of real graphs?"

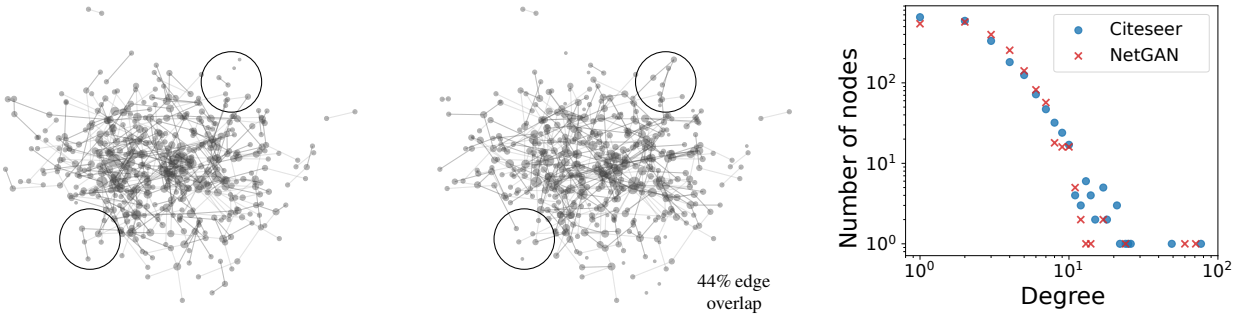
[*]Equal contribution [1]Technical University of Munich, Germany. Correspondence to: Daniel Zügner <zuegnerd@in.tum.de>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

An increasingly popular way to address this issue in other fields is by switching from *explicit* (prescribed) models to *implicit* ones. This transition is especially notable in computer vision, where generative adversarial networks (GANs) (Goodfellow et al., 2014) significantly advanced the state of the art over the classic prescribed approaches like mixtures of Gaussians (Blanken et al., 2007). GANs achieve unparalleled results in scenarios such as image and 3D objects generation (e.g., Karras et al., 2017; Berthelot et al., 2017; Wu et al., 2016). However, despite their massive success when dealing with real-valued data, adapting GANs to handle *discrete* objects like graphs or text remains an open research problem (Goodfellow, 2016). In fact, discreteness is only one of the obstacles when applying GANs to network data. Large repositories of graphs that all come from the same distribution are not available. This means that in a typical setting one has to learn from a *single graph*. Additionally, any model operating on a graph necessarily has to be *permutation invariant*, as graphs are isomorphic under node reordering.

In this work we introduce *NetGAN* – the first implicit generative model for graphs and networks that tackles all of the above challenges. We formulate the problem of learning the graph topology as learning the distribution of biased random walks over the graph. Like in the typical GAN setting, the generator G – in our case defined as a stochastic neural network with discrete output samples – learns to generate random walks that are *plausible* in the real graph, while the discriminator D then has to distinguish them from the true ones that are sampled from the original graph.

The main requirement for a graph generative model is the ability to generate realistic graphs. In the experimental section we compare NetGAN to other established prescribed models on this task. We observe that our proposed method consistently reproduces most known patterns inherent to real-world networks without explicitly specifying any of them in the model definition (e.g., degree distribution, as seen in Fig. 1). However, a model that simply replicates the original graph would also trivially fulfill this requirement, which clearly isn't our goal. In order to prove that this is not the case we examine the generalization properties of NetGAN by evaluating its link prediction performance. As our experiments show, our model exhibits competitive performance in this task and even achieves state-of-the-art



(a) Original graph (b) Graph generated by NetGAN (c) Degree distribution comparison

Figure 1: (a) Subgraph of the CITESEER network and (b) the respective subset of the graph generated by NetGAN. Both have similar structure but are not identical. (c) shows that the degree distributions of the two graphs are very close.

results on some datasets. This result is especially impressive, since NetGAN is not trained explicitly for performing link prediction. To summarize, our main contributions are:

- We introduce NetGAN[1] – the first of its kind GAN architecture that generates graphs via random walks. Our model tackles the associated challenges of staying permutation invariant, learning from a single graph and generating discrete output.
- We show that our method preserves important topological properties, without having to explicitly specifying them in the model definition. Moreover, we demonstrate how latent space interpolation leads to producing graphs with smoothly changing characteristics.
- We highlight the generalization properties of NetGAN by its link prediction performance that is competitive with the state of the art on real-word datasets, despite the model not being trained explicitly for this task.

2. Related Work

So far, no GAN architectures applicable to real-world networks have been proposed. Liu et al. (2017) propose a GAN architecture for learning topological features of subgraphs. Tavakoli et al. (2017) apply GANs to graph data by trying to directly generate adjacency matrices. Because their model produces the entire adjacency matrix – including the zero entries – it requires computations and memory quadratic in the number of nodes. Such quadratic complexity is infeasible in practice, allowing to process only small graphs, with reported runtime of over 60 hours for a graph with only 154 nodes. In contrast, NetGAN operates on random walks – it considers only the non-zero entries of the adjacency matrix efficiently exploiting the sparsity of real-world graphs – and is readily applicable to graphs with thousands of nodes.

[1] Code available at: https://www.kdd.in.tum.de/netgan

Deep learning methods for graph data have mostly been studied in the context of node embeddings (Perozzi et al., 2014; Grover & Leskovec, 2016; Kipf & Welling, 2016). The main idea behind these approaches is that of modeling the probabilities of each individual edge's existence, $p(A_{uv})$, as some function of the respective node embeddings, $f(\boldsymbol{h}_u, \boldsymbol{h}_v)$, where f is represented by a neural network. The recently proposed GraphGAN (Wang et al., 2017) is another instance of such prescribed edge-level probabilistic models, where f is optimized using the GAN objective instead of the traditional cross-entropy. Deep embedding based methods achieve state-of-the-art scores in tasks like link prediction and node classification. Nevertheless, as we show in Sec. 3.2, using such approaches for generating entire graphs produces samples that don't preserve any of the patterns inherent to real-world networks.

Prescribed generative models for graphs have a long history and are well-studied. For a survey we refer the reader to Chakrabarti & Faloutsos (2006) and Goldenberg et al. (2010). Typically, prescribed generative approaches are designed to capture and reproduce some predefined subset of graph properties (e.g., degree distribution, community structure, clustering coefficient). Notable examples include the configuration model (Bender & Canfield, 1978; Molloy & Reed, 1995), variants of the degree-corrected stochastic blockmodel (Karrer & Newman, 2011; Bojchevski & Günnemann), Exponential Random Graph Models (Holland & Leinhardt, 1981), Multiplicative Attribute Graph model (Kim & Leskovec, 2011), and the block two-level Erdős-Rényi random graph model (Seshadhri et al., 2012). In Sec. 4 we compare with some of these prescribed models on the tasks of graph generation and link prediction.

Due to the challenging nature of the problem, only few approaches able to generate discrete data using GANs exist. Most approaches focus on generating discrete sequences such as text, with some of them using reinforcement learn-

ing techniques to enable backpropagation through sampling discrete random variables (Yu et al., 2017; Kusner & Hernández-Lobato, 2016; Li et al., 2017; Liang et al., 2017). Other approaches modify the GAN objective to tackle the same challenge (Che et al., 2017; Hjelm et al., 2017). Focusing on non-sequential discrete data, Choi et al. (2017) generate high-dimensional discrete features (e.g. binary indicators, counts) in patient records. None of these methods have considered graph structured data.

3. Model

In this section we introduce NetGAN - a Generative Adversarial Network model for graph / network data. Its core idea lies in capturing the topology of a graph by learning a distribution over the random walks. Given an input graph of N nodes, defined by a binary adjacency matrix $\boldsymbol{A} \in \{0,1\}^{N\times N}$, we first sample a set of random walks of length T from $\boldsymbol{A}$. This collection of random walks serves as a training set for our model. We use the biased second-order random walk sampling strategy described in Grover & Leskovec (2016), as it better captures both local and global graph structure. An important advantage of using random walks is their invariance under node reordering. Additionally, random walks only include the nonzero entries of $\boldsymbol{A}$, thus efficiently exploiting the sparsity of real-world graphs.

Like any typical GAN architecture, NetGAN consists of two main components - a generator G and a discriminator D. The goal of the generator is to generate synthetic random walks that are plausible in the input graph. At the same time, the discriminator learns to distinguish the synthetic random walks from the real ones that come from the training set. Both G and D are trained end-to-end using backpropagation. At any point of the training process it is possible to use G to generate a set of random walks, which can then be used to produce an adjacency matrix of a new generated graph. In the rest of this section we describe each stage of this process and our design choices in more detail. An overview of our model's complete architecture can be seen in Fig. 2.

3.1. Architecture

Generator. The generator G defines an implicit probabilistic model for generating random walks: $(\boldsymbol{v}_1, ..., \boldsymbol{v}_T) \sim G$. We model G as a sequential process based on a neural network f_θ parametrized by θ. At each step t, f_θ produces two values: the probability distribution over the next node to be sampled, parametrized by the logits $\boldsymbol{p}_t$, and the current memory state of the model, denoted as $\boldsymbol{m}_t$. The next node $\boldsymbol{v}_t$, represented as a one-hot vector, is sampled from a categorical distribution $\boldsymbol{v}_t \sim \mathrm{Cat}(\sigma(\boldsymbol{p}_t))$, where $\sigma(\cdot)$ denotes the softmax function, and together with $\boldsymbol{m}_t$ is passed into f_θ at the next step $t+1$. Similarly to the classic GAN setting, a latent code $\boldsymbol{z}$ drawn from a multivariate standard normal distribution is passed through a parametric function $g_{\theta'}$ to initialize $\boldsymbol{m}_0$. The generative process of G is summarized in the box below.

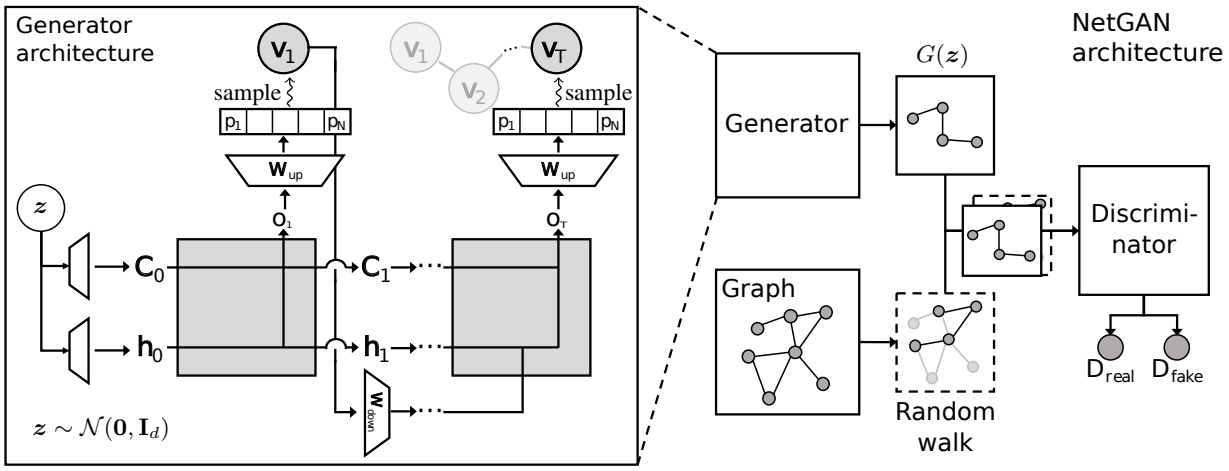
$$
\begin{aligned}
& & \boldsymbol{z} &\sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_d) \\
& & \boldsymbol{m}_0 &= g_{\theta'}(\boldsymbol{z}) \\
\boldsymbol{v}_1 &\sim \mathrm{Cat}(\sigma(\boldsymbol{p}_1)), & (\boldsymbol{p}_1, \boldsymbol{m}_1) &= f_\theta(\boldsymbol{m}_0, \boldsymbol{0}) \\
\boldsymbol{v}_2 &\sim \mathrm{Cat}(\sigma(\boldsymbol{p}_2)), & (\boldsymbol{p}_2, \boldsymbol{m}_2) &= f_\theta(\boldsymbol{m}_1, \boldsymbol{v}_1) \\
& \vdots & & \vdots \\
\boldsymbol{v}_T &\sim \mathrm{Cat}(\sigma(\boldsymbol{p}_T)), & (\boldsymbol{p}_T, \boldsymbol{m}_T) &= f_\theta(\boldsymbol{m}_{T-1}, \boldsymbol{v}_{T-1})
\end{aligned}
$$

In this work we focus our attention on the Long short-term memory (LSTM) architecture for f_θ, introduced by Hochreiter & Schmidhuber (1997). The memory state $\boldsymbol{m}_t$ of an LSTM is represented by the cell state $\boldsymbol{C}_t$, and the hidden state $\boldsymbol{h}_t$. The latent code $\boldsymbol{z}$ goes through two separate streams, each consisting of two fully connected layers with $\tanh$ activation, and then used to initialize $(\boldsymbol{C}_0, \boldsymbol{h}_0)$.

A natural question might arise: "Why use a model with memory and temporal dependencies, when the random walks are Markov processes?" (2nd order Markov for biased RWs). Or put differently, what's the benefit of using random walks of length greater than 2? In theory, a model with large enough capacity could simply memorize all existing edges in the graph and recreate them. However, for large graphs achieving this in practice is not feasible. More importantly, pure memorization is not the goal of NetGAN, rather we want to have generalization and to generate graphs with similar properties, not exact replicas. Having longer random walks combined with memory helps the model to learn the topology and general patterns in the data (e.g., community structure). Our experiments in Sec. 4.2 confirm this, showing that longer random walks are indeed beneficial.

After each time step, to generate the next node in the random walk, the network f_θ should output the logits $\boldsymbol{p}_t$ of length N. However, operating in such high dimensional space leads to an unnecessary computational overhead. To tackle this issue, the LSTM outputs $\boldsymbol{o}_t \in \mathbb{R}^H$ instead, with $H \ll N$, which is then up-projected to $\mathbb{R}^N$ using the matrix $\boldsymbol{W}_{up} \in \mathbb{R}^{H\times N}$. This enables us to efficiently handle large-scale graphs.

Sampling the next node in the random walk $\boldsymbol{v}_t$ presents another challenge. Since sampling from a categorical distribution is a non-differentiable operation it blocks the flow of gradients and precludes backpropagation. We solve this problem by using the Straight-Through Gumbel estimator by Jang et al. (2016). More specifically, we perform the following transformation: First, we let $\boldsymbol{v}_t^* = \sigma\left((\boldsymbol{p}_t + \boldsymbol{g})/\tau\right)$, where τ is a temperature parameter, and g_i's are i.i.d. samples from a Gumbel distribution with zero mean and unit scale. Then, the next sample is



(a) Generator architecture (b) NetGAN architecture

Figure 2: The NetGAN architecture proposed in this work (b) and the generator architecture (a).

chosen as $\boldsymbol{v}_t = \text{onehot}(\arg\max \boldsymbol{v}_t^*)$. While the one-hot sample $\boldsymbol{v}_t$ is passed as input to the next time step, during the backward pass the gradients will flow through the differentiable $\boldsymbol{v}_t^*$. The choice of τ allows to trade-off between better flow of gradients (large τ, more uniform $\boldsymbol{v}_t^*$) and more exact calculations (small τ, $\boldsymbol{v}_t^* \approx \boldsymbol{v}_t$).

Now that a new node $\boldsymbol{v}_t$ is sampled, it needs to be projected back to a lower-dimensional representation before feeding into the LSTM. This is done by means of down-projection matrix $\boldsymbol{W}_{down} \in \mathbb{R}^{N \times H}$.

Discriminator. The discriminator D is based on the standard LSTM architecture. At every time step t, a one-hot vector $\boldsymbol{v}_t$, denoting the node at the current position, is fed as input. After processing the entire sequence of T nodes, the discriminator outputs a single score that represents the probability of the random walk being real.

3.2. Training

Wasserstein GAN. We train our model based on the Wasserstein GAN (WGAN) framework (Arjovsky et al., 2017), as it prevents mode collapse and leads to more stable training overall. To enforce the Lipschitz constraint of the discriminator, we use the gradient penalty as in Gulrajani et al. (2017). The model parameters $\{\theta, \theta'\}$ are trained using stochastic gradient descent with Adam (Kingma & Ba, 2014). Weights are regularized with an L_2 penalty.

Early stopping. Because we are interested in generalizing the input graph, the "trivial" solution where the generator has memorized all existing edges is of no interest to us. This means that we need to control how closely the generated graphs resemble the original one. To achieve this, we propose two possible early stopping strategies, either of which can be used depending on the task at hand. The first strategy, named VAL-CRITERION is concerned with the generalization properties of NetGAN. During training, we keep a sliding window of the random walks generated in the last 1,000 iterations and use them to construct a matrix of transition counts. This matrix is then used to evaluate the link prediction performance on a validation set (i.e. ROC and AP scores, for more details see Sec. 4.2). We stop with training when the validation performance stops improving.

The second strategy, named EO-CRITERION makes NetGAN very flexible and gives the user control over the graph generation. We stop training when we achieve a user specified edge overlap between the generated graphs (see next section) and the original one at a given iteration. Based on her end task the user can choose to generate graphs with either small or large edge overlap with the original, while maintaining structural similarity. This will lead to generated graphs that either generalize better or are closer replicas respectively, yet still capture the properties of the original.

3.3. Assembling the Adjacency Matrix

After finishing the training, we use the generator G to construct a score matrix $\boldsymbol{S}$ of transition counts, i.e. we count how often an edge appears in the set of generated random walks (typically, using a much larger number of random walks than for early stopping, e.g., 500K). While the raw counts matrix $\boldsymbol{S}$ is sufficient for link prediction purposes, we need to convert it to a binary adjacency matrix $\hat{\boldsymbol{A}}$ if we wish to reason about the synthetic graph. First, $\boldsymbol{S}$ is symmetrized by setting $s_{ij} = s_{ji} = \max\{s_{ij}, s_{ji}\}$. Because we cannot explicitly control the starting node of the random walks generated by G, some high-degree nodes will likely be overrepresented. Thus, a simple binarization strategy like thresholding or choosing top-k entries might lead to leaving out the low-degree nodes and producing singletons.

Table 1: Statistics of CORA-ML and the graphs generated by NetGAN and the baselines, averaged over 5 trials. NetGAN closely matches the input networks in most properties, while other methods either deviate significantly in at least one statistic or overfit. * indicates values for the conf. model that by definition exactly match the original.

Graph		Max. degree	Assort-ativity	Triangle count	Power law exp.	Inter-comm. unity density	Intra-comm. unity density	Cluster-ing coeff.	Charac. path len.	Average rank
CORA-ML		240	-0.075	2,814	1.860	4.3e-4	1.7e-3	2.73e-3	5.61	
Conf. model	(1% EO)	*	-0.030	322	*	1.6e-3	2.8e-4	3.00e-4	4.38	7.50
Conf. model	(52% EO)	*	-0.051	626	*	9.8e-4	9.9e-4	6.10e-4	4.46	5.83
DC-SBM	(11% EO)	165	-0.052	1,403	1.814	6.7e-4	1.2e-3	3.30e-3	5.12	3.36
ERGM	(56% EO)	243	-0.077	2,293	1.786	6.9e-4	1.2e-3	2.17e-3	4.59	2.88
BTER	(2.2% EO)	199	0.033	3,060	1.787	1.0e-3	7.5e-4	4.62e-3	4.59	4.75
VGAE	(0.3% EO)	13	-0.009	14	1.674	1.4e-3	3.2e-4	1.17e-3	5.28	5.88
NetGAN VAL	(39% EO)	199	-0.060	1,410	1.773	6.5e-4	1.3e-3	2.33e-3	5.17	3.00
NetGAN EO	(52% EO)	233	-0.066	1,588	1.793	6.0e-4	1.4e-3	2.44e-3	5.20	1.75

To address this issue, we use the following approach: (i) We ensure that every node i has at least one edge by sampling a neighbor j with probability $p_{ij} = \frac{s_{ij}}{\sum_v s_{iv}}$. If an edge was already sampled before, we repeat the procedure; (ii) We continue sampling edges without replacement using for each edge (i, j) the probability $p_{ij} = \frac{s_{ij}}{\sum_{u,v} s_{uv}}$, until we reach the desired amount of edges (e.g., as many edges as in the original graph). To obtain an undirected graph for every edge (i, j) we also include (j, i). Note that this procedure is not guaranteed to produce a fully connected graph.

4. Experiments

In this section we evaluate the quality of the graphs generated by NetGAN by computing various graph statistics. We quantify the generalization power of the proposed model by evaluating its link prediction performance. Furthermore, we demonstrate how we can generate graphs with smoothly changing properties via latent space interpolation. Additional experiments are provided in the supp. mat.

Datasets. For the experiments we use five well-known citation datasets and the Political Blogs dataset. For the large CORA dataset and its commonly used subset of machine learning papers denoted with CORA-ML we use the same preprocessing as in Bojchevski & Günnemann (2018). For all the experiments we treat the graphs as undirected and only consider the largest connected component (LCC). Information about the datasets is listed in Table 2.

Table 2: Dataset statistics. $\mathbf{N_{LCC}}, \mathbf{E_{LCC}}$ - number of nodes and edges respectively in the largest connected component.

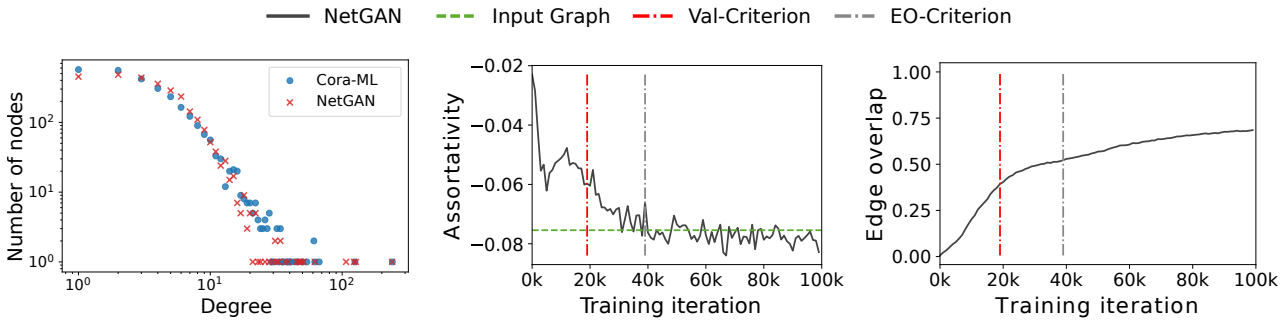
Name	$\mathbf{N_{LCC}}$	$\mathbf{E_{LCC}}$	Reference
CORA-ML	2,810	7,981	(McCallum et al., 2000)
CORA	18,800	64,529	(McCallum et al., 2000)
CITESEER	2,110	3,757	(Sen et al., 2008)
PUBMED	19,717	44,324	(Sen et al., 2008)
DBLP	16,191	51,913	(Pan et al., 2016)
POL. BLOGS	1,222	16,714	(Adamic & Glance, 2005)

4.1. Graph Generation

Setup. In this task, we fit NetGAN to the CORA-ML and CITESEER citation networks in order to evaluate quality of the generated graphs. We compare to the following baselines: configuration model (Molloy & Reed, 1995), degree-corrected stochastic blockmodel (DC-SBM) (Karrer & Newman, 2011), exponential random graph model (ERGM) (Holland & Leinhardt, 1981) and the block two-level Erdős-Rényi random graph model (BTER) (Seshadhri et al., 2012). Additionally, we use the variational graph autoencoder (VGAE) (Kipf & Welling, 2016) as a representative of network embedding approaches. We randomly hide 15% of the edges (which are used for the stopping criterion; see Sec. 3.2) and fit all the models on the remaining graph. We sample 5 graphs from each of the trained models and report their average statistics in Table 1. Definitions of the statistics, additional metrics, standard deviations and details about the baselines are given in the supplementary material.

Evaluation. The general trend that becomes apparent from the results in Table 1 (and Table 2 in supplementary material) is that prescribed models excel at recovering the statistics that they directly model (e.g., degree sequence for DC-SBM). At the same time, these models struggle when dealing with graph properties that they don't account for (e.g., assortativity for BTER). On the other hand, NetGAN is able to capture all the graph properties well, although none of them are explicitly specified in its model definition. We also see that VGAE is not able to produce realistic graphs. This is expected, since the main purpose of VGAE is learning node embeddings, and not generating entire graphs.

The final column shows the average rank of each method for all statistics, with NetGAN performing the best. ERGM seems to be performing surprisingly well, however it suffers from severe overfitting – using the same fitted ERGM for the link prediction task we get both AUC and AP scores close to 0.5 (worst possible value). In contrast, NetGAN does a good job both at preserving properties in generated graphs, as well as generalizing, as we see in Sec. 4.2.



(a) Degree distribution

(b) Assortativity over training iterations

(c) Edge overlap (EO) over training iterations

Figure 3: Properties of graphs generated by NetGAN trained on CORA-ML.

Is the good performance of NetGAN in this experiment only due to the overlapping edges (existing in the input graph)? To rule out this possibility we perform the following experiment: We take the graph generated by NetGAN, fix the overlapping edges and rewire the rest according to the configuration model. The properties of the resulting graph (row #3 in Table 1) deviate strongly from the input graph. This confirms that NetGAN does not simply memorize some edges and generates the rest at random, but rather captures the underlying structure of the network.

In line with our intuition, we can see that higher EO leads to generated graphs with statistics closer to the original. Figs. 3b and 3c show how the graph statistics evolve during the training process. Fig. 3c shows that the edge overlap smoothly increasing with the number of epochs. We provide plots for other statistics and for CITESEER in the supp. mat.

4.2. Link Prediction

Setup. Link prediction is a common graph mining task where the goal is to predict the existence of unobserved links in a given graph. We use it to evaluate the generalization properties of NetGAN. We hold out 10% of edges from the graph for validation and 5% as the test set, along with the same amount of randomly selected non-edges, while ensuring that the training network remains connected. We measure the performance with two commonly used metrics: area under the ROC curve (AUC) and average precision (AP). To evaluate NetGAN's performance, we sample a given number of random walks (500K/100M) from the trained generator and we use the observed transition counts between any two nodes as a measure of how likely there is an edge between them. We compare with DC-SBM, node2vec and VGAE, as well as Adamic/Adar(Adamic & Adar, 2003).

Evaluation. The results are listed in Table 3. There is no overall dominant method, with different methods achieving best results on different datasets. NetGAN shows competitive performance for all datasets, even achieving state-of-the-art results for some of them (CITESEER and POLBLOGS), despite not being explicitly trained for this task.

Interestingly, the NetGAN performance increases when increasing the number of random walks sampled from the generator. This is especially true for the larger networks (CORA, DBLP, PUBMED), since given their size we need more random walks to cover the entire graph. This suggests that for an additional computational cost one can get significant gains in link prediction performance. Note, that while 100M may seem like a large number, the sampling procedure can be trivially parallelized.

Sensitivity analysis. Although NetGAN has many hyperparameters – typical for a GAN model – in practice most of them are not critical for performance, as long as they are within a reasonable range (e.g. $H \geq 30$). One important exception is the the random walk length T. To choose the optimal value, we evaluate the change in link prediction performance as we vary T on CORA-ML. We train multiple models with different random walk lengths, and evaluate the scores ensuring each one observes equal number of transitions. Results averaged over 5 runs are given in Fig. 6. We empirically confirm that the model benefits from using longer random walks as opposed to just edges (i.e. T=2). The performance gain for $T = 20$ over $T = 16$ is marginal and does not outweigh the additional computational cost, thus we set $T = 16$ for all experiments.

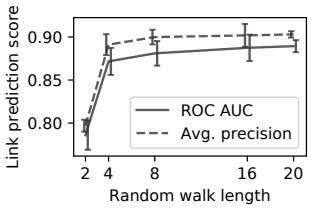


Figure 6: Effect of the random walk length T on the performance.

Table 3: Link prediction performance (in %).

Method	CORA-ML		CORA		CITESEER		DBLP		PUBMED		POLBLOGS	
	AUC	AP	AUC	AP	AUC	AP	AUC	AP	AUC	AP	AUC	AP
Adamic/Adar	92.16	85.43	93.00	86.18	88.69	77.82	91.13	82.48	84.98	70.14	85.43	92.16
DC-SBM	**96.03**	95.15	98.01	97.45	94.77	93.13	**97.05**	96.57	**96.76**	95.64	95.46	**94.93**
node2vec	92.19	91.76	**98.52**	**98.36**	95.29	94.58	96.41	96.36	96.49	95.97	85.10	83.54
VGAE	95.79	**96.30**	97.59	97.93	95.11	96.31	96.38	**96.93**	94.50	**96.00**	93.73	94.12
NetGAN (500K)	94.00	92.32	82.31	68.47	95.18	91.93	82.45	70.28	87.39	76.55	95.06	94.61
NetGAN (100M)	95.19	95.24	84.82	88.04	**96.30**	**96.89**	86.61	89.21	93.41	94.59	**95.51**	94.83

4.3. Latent Variable Interpolation

Setup. Latent space interpolation is a good way to gain insight into what kind of structure the generator was able to capture. To be able to visualize the properties of the generated graphs we train our model using a 2-dimensional noise vector $\boldsymbol{z}$ drawn as before from a bivariate standard normal distribution. This corresponds to a 2-dimensional latent space $\Omega = \mathbb{R}^2$. Then, instead of sampling $\boldsymbol{z}$ from the entire latent space Ω, we now sample from subregions of Ω and visualize the results. Specifically, we divide Ω into 20×20 subregions (bins) of equal probability mass using the standard normal cumulative distribution function Φ. For each bin we generate 62.5K random walks. We evaluate properties of both the generated *random walks* themselves, as well as properties of the resulting *graphs* obtained by sampling a binary adjacency matrix for each bin.

Evaluation. In Fig. 4a and 4b we see properties of the generated random walks; in Fig. 4c and 4d, we visualize properties of graphs sampled from the random walks in the respective bins. In all four heatmaps, we see distinct patterns, e.g. higher average degree of starting nodes for the bottom right region of Fig. 4a, or higher degree distribution inequality in the top-right area of Fig. 4c. While Fig. 4c and 4d show that certain regions of $\boldsymbol{z}$ correspond to generated graphs with very different degree distributions, recall that sampling from the entire latent space (Ω) yields graphs with degree distribution similar to the original graph (see Fig. 1c). The model was trained on CORA-ML. More heatmaps for other metrics (16 in total) and visualizations for CITESEER can be found in the supplementary material.

This experiment clearly demonstrates that by interpolating in the latent space we can obtain graphs with smoothly changing properties. The smooth transitions in the heatmaps provide evidence that our model learns to map specific parts of the latent space to specific properties of the graph.

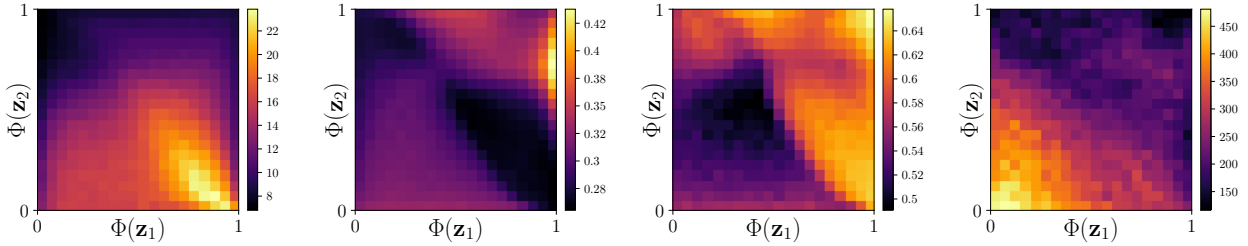
We can also see this mapping from latent space to the generated graph properties in the community distribution histograms on a 10×10 grid in Fig. 5. Marked by (*) and (Ω) we see the community distributions for the input graph and the graph obtained by sampling on the complete latent space respectively. In Fig. 5b and 5c, we see the evolution of selected community shares when following a trajectory from top to bottom, and left to right, respectively. The community histograms resulting from sampling random walks from opposing regions of the latent space are very different; again the transitions between these histograms are smooth, as can be seen in the trajectories in Fig. 5b and 5c.

5. Discussion and Future Work

When evaluating different graph generative models in Sec. 3.2, we observed a major limitation of explicit models. While the prescribed approaches excel at recovering the properties directly included in their definition, they perform significantly worse with respect to the rest. This clearly indicates the need for implicit graph generators such as NetGAN. Indeed, we notice that our model is able to consistently capture all the important graph characteristics (see Table 1). Moreover, NetGAN generalizes beyond the input graph, as can be seen by its strong link prediction performance in Sec. 4.2. Still, being the first model of its kind, NetGAN possesses certain limitations, and a number of related questions could be addressed in follow-up works:

Scalability. We have observed in Sec. 4.2 that it takes a large number of generated random walks to get representative transition counts for large graphs. While sampling random walks from NetGAN is trivially parallelizable, a possible extension of our model is to use a *conditional* generator, i.e. the generator can be provided a desired starting node, thus ensuring a more even coverage. On the other hand, the sampling procedure itself can be sped up by incorporating a hierarchical softmax output layer - a method commonly used in natural language processing.

Evaluation. It is nearly impossible to judge whether a graph is realistic by visually inspecting it (unlike images, for example). In this work we already quantitatively evaluate the performance of NetGAN on a large number of standard graph statistics. However, developing new measures applicable to (implicit) graph generative models will deepen our understanding of their behavior, and is an important direction for future work.



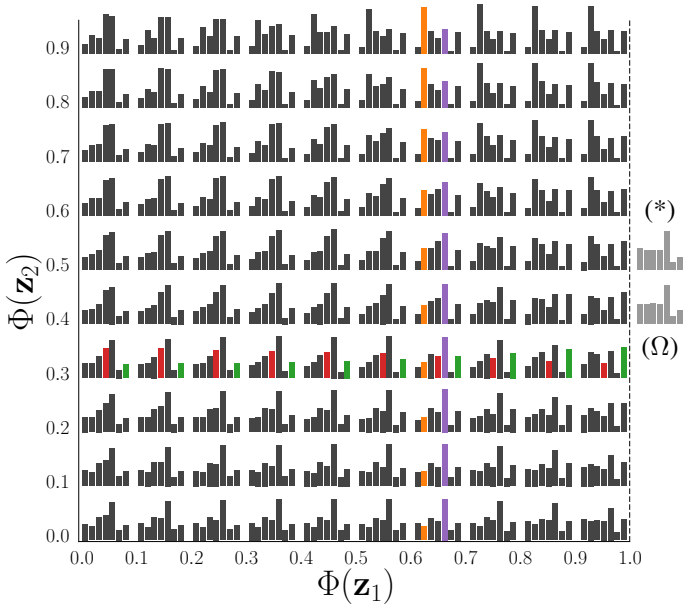
(a) Avg. degree of start node

(b) Avg. share of nodes in start community

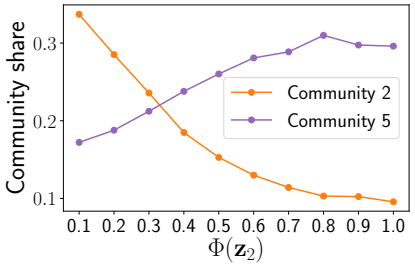
(c) Gini coefficient (input graph: 0.48)

(d) Max. degree (input graph: 240)

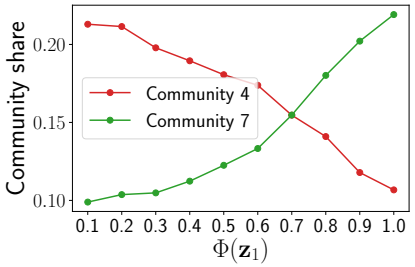
Figure 4: Properties of the random walks (4a and 4b) as well as the graphs (4c and 4d) sampled from the 20×20 bins.



(a) Community histograms



(b) Top to bottom trajectory



(c) Left to right trajectory

Figure 5: Community histograms of graphs sampled from subsets of the latent space. (a) shows complete community histograms on a 10×10 grid. (b) and (c) show how shares of specific communities change along trajectories. (Ω) is the community distribution when sampling from the entire latent space, and (*) is the community histogram of CORA-ML. Available as an animation at https://goo.gl/bkNcVa.

Experimental scope. In the current work we focus on the setting of a single large graph. Adaptation to other scenarios, such as a collection of smaller i.i.d. graphs, that frequently occur in other fields (e.g., chemistry, biology), would be an important extension of our model. Studying the influence of the graph topology (e.g., sparsity, diameter) on NetGAN's performance will shed more light on the model's properties.

Other types of graphs. While plain graphs are ubiquitous, many of important applications deal with attributed, k-partite or heterogeneous networks. Adapting the NetGAN model to handle these other modalities of the data is a promising direction for future research. Especially important would be an adaptation to the dynamic / inductive setting, where new nodes are added over time.

6. Conclusion

In this work we introduce NetGAN - an implicit generative model for network data. NetGAN is able to generate graphs that capture important topological properties of complex networks, such as community structure and degree distribution, without having to manually specify any of them. Moreover, our proposed model shows strong generalization properties, as highlighted by its competitive link prediction performance on a number of datasets. NetGAN can also be used for generating graphs with continuously varying characteristics using latent space interpolation. Combined our results provide strong evidence that implicit generative models for graphs are well-suited for capturing the complex nature of real-world networks.

Acknowledgments

This research was supported by the German Research Foundation, Emmy Noether grant GU 1409/2-1, and by the Technical University of Munich - Institute for Advanced Study, funded by the German Excellence Initiative and the European Union Seventh Framework Programme under grant agreement no 291763, co-funded by the European Union.

References

Adamic, L. A. and Adar, E. Friends and neighbors on the web. *Social networks*, 25(3):211–230, 2003.

Adamic, L. A. and Glance, N. The political blogosphere and the 2004 US election: divided they blog. In *Proceedings of the international workshop on Link discovery*, pp. 36–43, 2005.

Arjovsky, M., Chintala, S., and Bottou, L. Wasserstein GAN. *arXiv preprint arXiv:1701.07875*, 2017.

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

Bender, E. A. and Canfield, E. R. The asymptotic number of labeled graphs with given degree sequences. *Journal of Combinatorial Theory, Series A*, 24(3):296–307, 1978.

Berthelot, D., Schumm, T., and Metz, L. Began: Boundary equilibrium generative adversarial networks. *arXiv preprint arXiv:1703.10717*, 2017.

Blanken, H. M., de Vries, A. P., Blok, H. E., and Feng, L. *Multimedia retrieval*. 2007.

Bojchevski, A. and Günnemann, S. Bayesian robust attributed graph clustering: Joint learning of partial anomalies and group structure. In *Proceedings of the AAAI Conference on Artificial Intelligence, 2018*, pp. 2738–2745.

Bojchevski, A. and Günnemann, S. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. In *International Conference on Learning Representations*, 2018.

Broido, A. D. and Clauset, A. Scale-free networks are rare. *arXiv preprint arXiv:1801.03400*, 2018.

Chakrabarti, D. and Faloutsos, C. Graph mining: Laws, generators, and algorithms. *Computing Surveys (CSUR)*, 38(1):2, 2006.

Che, T., Li, Y., Zhang, R., Hjelm, R. D., Li, W., Song, Y., and Bengio, Y. Maximum-likelihood augmented discrete generative adversarial networks. *arXiv preprint arXiv:1702.07983*, 2017.

Choi, E., Biswal, S., Malin, B., Duke, J., Stewart, W. F., and Sun, J. Generating multi-label discrete electronic health records using generative adversarial networks. *arXiv preprint arXiv:1703.06490*, 2017.

Dong, Y., Johnson, R. A., Xu, J., and Chawla, N. V. Structural diversity and homophily: A study across more than one hundred big networks. In *Proceedings of the SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 807–816, 2017.

Goldenberg, A., Zheng, A. X., Fienberg, S. E., Airoldi, E. M., et al. A survey of statistical network models. *Foundations and Trends in Machine Learning*, 2(2):129–233, 2010.

Goodfellow, I. NIPS 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

Grover, A. and Leskovec, J. node2vec: Scalable feature learning for networks. In *Proceedings of the SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864, 2016.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. Improved training of Wasserstein GANs. *arXiv preprint arXiv:1704.00028*, 2017.

Handcock, M. S., Hunter, D. R., Butts, C. T., Goodreau, S. M., Krivitsky, P. N., and Morris, M. *ERGM: Fit, Simulate and Diagnose Exponential-Family Models for Networks*. The Statnet Project, 2017. R package version 3.8.0.

Hjelm, R. D., Jacob, A. P., Che, T., Cho, K., and Bengio, Y. Boundary-seeking generative adversarial networks. *arXiv preprint arXiv:1702.08431*, 2017.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Holland, P. W. and Leinhardt, S. An exponential family of probability distributions for directed graphs. *Journal of the american Statistical association*, 76(373):33–50, 1981.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with Gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Karras, T., Aila, T., Laine, S., and Lehtinen, J. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

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

Kim, M. and Leskovec, J. Modeling social networks with node attributes using the multiplicative attribute graph model. *arXiv preprint arXiv:1106.5053*, 2011.

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

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

Kusner, M. J. and Hernández-Lobato, J. M. GANs for sequences of discrete elements with the Gumbel-softmax distribution. *arXiv preprint arXiv:1611.04051*, 2016.

Li, J., Monroe, W., Shi, T., Ritter, A., and Jurafsky, D. Adversarial learning for neural dialogue generation. *arXiv preprint arXiv:1701.06547*, 2017.

Liang, X., Hu, Z., Zhang, H., Gan, C., and Xing, E. P. Recurrent topic-transition GAN for visual paragraph generation. *arXiv preprint arXiv:1703.07022*, 2017.

Liu, W., Chen, P.-Y., Cooper, H., Oh, M. H., Yeung, S., and Suzumura, T. Can GAN learn topological features of a graph? *arXiv preprint arXiv:1707.06197*, 2017.

McCallum, A. K., Nigam, K., Rennie, J., and Seymore, K. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 2000.

Molloy, M. and Reed, B. A critical point for random graphs with a given degree sequence. *Random structures & algorithms*, 6(2-3):161–180, 1995.

Pan, S., Wu, J., Zhu, X., Zhang, C., and Wang, Y. Tri-party deep network representation. *Network*, 11(9):12, 2016.

Peixoto, T. P. The graph-tool python library. URL http://figshare.com/articles/graph_tool/1164194.

Perozzi, B., Al-Rfou, R., and Skiena, S. Deepwalk: Online learning of social representations. In *Proceedings of the SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710, 2014.

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

Seshadhri, C., Kolda, T. G., and Pinar, A. Community structure and scale-free collections of Erdős-Rényi graphs. *Physical Review E*, 85(5):056109, 2012.

Tavakoli, S., Hajibagheri, A., and Sukthankar, G. Learning social graph topologies using generative adversarial neural networks. 2017.

Wang, H., Wang, J., Wang, J., Zhao, M., Zhang, W., Zhang, F., Xie, X., and Guo, M. GraphGAN: Graph representation learning with generative adversarial nets. *arXiv preprint arXiv:1711.08267*, 2017.

Wu, J., Zhang, C., Xue, T., Freeman, B., and Tenenbaum, J. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in Neural Information Processing Systems*, pp. 82–90, 2016.

Yu, L., Zhang, W., Wang, J., and Yu, Y. SeqGAN: Sequence generative adversarial nets with policy gradient. In *AAAI*, pp. 2852–2858, 2017.

A. Graph statistics

Table 4: Graph statistics used to measure graph properties in this work.

Metric name	Computation	Description
Maximum degree	$\max_{v \in V} d(v)$	Maximum degree of all nodes in a graph.
Assortativity	$\rho = \frac{\mathrm{cov}(X,Y)}{\sigma_X \sigma_Y}$	Pearson correlation of degrees of connected nodes, where the (x_i, y_i) pairs are the degrees of connected nodes.
Triangle count	$\frac{\lvert\{\{u,v,w\} \lvert \{(u,v),(v,w),(u,w)\} \subseteq E\}\rvert}{6}$	Number of triangles in the graph, where $u \sim v$ denotes that u and v are connected.
Power law exponent	$1 + n \left(\sum_{u \in V} \log \frac{d(u)}{d_{\min}} \right)^{-1}$	Exponent of the power law distribution, where d_{min} denotes the minimum degree in a network.
Inter-community density	$\frac{1}{K} \sum_{j=1}^{K} \sum_{\substack{k=1 \\ k \neq j}}^{K} \frac{1}{\lvert C_k \rvert \lvert C_j \rvert} \sum_{u \in C_j} \sum_{v \in C_k} A_{uv}$	Fraction of possible inter-community edges present in graph.
Intra-community density	$\frac{1}{K} \sum_{j=1}^{K} \frac{1}{\binom{\lvert C_j \rvert}{2}} \sum_{u,v \in C_j} A_{uv}$	Fraction of possible intra-community edges present in graph.
Wedge count	$\sum_{v \in V} \binom{d(v)}{2}$	Number of wedges (2-stars), i.e. two-hop paths in an undirected graph.
Rel. edge distr. entropy	$\frac{1}{\ln \lvert V \rvert} \sum_{v \in V} -\frac{d(v)}{\lvert E \rvert} \ln \frac{d(v)}{\lvert E \rvert}$	Entropy of degree distribution, 1 means uniform, 0 means a single node is connected to all others.
LCC	$N_{max} = \max_{f \subseteq F} \lvert f \rvert$	Size of largest connected component, where F are all connected components of the graph.
Claw count	$\sum_{v \in V} \binom{d(v)}{3}$	Number of claws (3-stars)
Gini coefficient	$\frac{2 \sum_{i=1}^{\lvert V \rvert} i \hat{d}_i}{\lvert V \rvert \sum_{i=1}^{\lvert V \rvert} \hat{d}_i} - \frac{\lvert V \rvert + 1}{\lvert V \rvert}$	Common measure for inequality in a distribution, where $\hat{\boldsymbol{d}}$ is the sorted list of degrees in the graph.
Community distribution	$c_i = \frac{\sum_{v \in C_i} d(v)}{\sum_{v \in V} d(v)}$	Share of in- and outgoing edges of community C_i, normalized by the number of edges in the graph.

B. Baselines

- **Configuration model.** In addition to randomly rewiring *all* edges in the input graph, we also generate random graphs with similar overlap as graphs generated by NetGAN using the configuration model. For this, we randomly select a share of edges (e.g. 39%) and keep them fixed, and shuffle the remaining edges. This leads to a graph with the specified edge overlap; in Table 2 we show that with the same edge overlap, NetGAN's generated graphs in general match the input graph better w.r.t the statistics we measure.

- **Exponential random graph model.** We use the R implementation of ERGM from the `ergm` package (Handcock et al., 2017). We used the following parameter settings: `edge count, density, degree correlation, deg1.5, and gwesp`. Here, `deg1.5` is the sum of all degrees to the power of 1.5, and `gwesp` refers to the geometrically weighted edgewise shared partner distribution.

- **Degree-corrected stochastic blockmodel.** We use the Python implementation from the `graph-tool` package (Peixoto) using the recommended hyperparameter settings.

- **Variational graph autoencoder.** We use the implementation provided by the authors (`https://github.com/tkipf/gae`). We construct the graph from the predicted edge probabilities using the same protocol as in Sec. 3.3 of our paper. To ensure a fair comparison we perform early stopping, i.e. select the weights that achieve the best validation set performance.

C. Properties of generated graphs

Table 5: Comparison of graph statistics between the CITESEER/CORA-ML graph and graphs generated by GraphGAN and DC-SBM, averaged after 5 trials.

Graph		Max. degree		Assortativity		Triangle count		Power law exponent		Avg. Inter-community density		Avg. Intra-community density		Clustering coefficient	
		Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.
CITESEER		77		-0.022		451		2.239		4.9e-4		9.3e-4		1.08e-2	
Conf. model		*	*	-0.017	± 0.006	20	± 6.50	*	*	1.1e-3	± 1e-5	2.3e-4	± 2e-5	5.80e-4	± 1.29e-4
Conf. model	(42% EO)	*	*	-0.020	± 0.009	54	± 8.8	*	*	8.4e-4	± 1e-5	5.1e-4	± 1e-5	1.33e-3	± 6.15e-5
Conf. model	(76% EO)	*	*	-0.024	± 0.006	207	± 11.8	*	*	6.3e-4	± 1e-5	7.6e-4	± 1e-5	5.00e-3	± 2.57e-4
DC-SBM	(6.6% EO)	53	± 5.6	0.022	± 0.018	257	± 30.9	2.066	± 0.014	7.6e-4	± 2e-5	5.3e-4	± 3e-5	1.00e-2	± 2.63e-3
ERGM	(27% EO)	66	± 1	0.052	± 0.005	415.6	± 8	2.0	± 0.01	9.3e-4	± 2e-5	4.8e-4	± 6e-6	1.49e-2	± 5.68e-4
BTER	(2% EO)	70	± 7.2	0.065	± 0.014	449	± 33	2.049	± 0.01	1.1e-3	± 2e-5	2.8e-4	± 6e-6	1.22e-2	± 2.31e-3
VGAe	(0.2% EO)	9.2	± 0.7	-0.057	± 0.016	2	± 1	2.039	± 0.00	1.2e-3	± 1e-5	2.5e-4	± 2e-5	1.35e-3	± 9.96e-4
NetGAN VAL	(42% EO)	54	± 4.2	-0.082	± 0.009	316	± 11.2	2.154	± 0.003	6.5e-4	± 2e-5	8.0e-4	± 2e-5	1.99e-2	± 3.48e-3
NetGAN EO	(76% EO)	63	± 4.3	-0.054	± 0.006	227	± 13.3	2.204	± 0.003	5.9e-4	± 2e-5	8.6e-4	± 1e-5	7.71e-3	± 2.43e-4
CORA-ML		240		-0.075		2,814		1.86		4.3e-4		1.7e-3		2.73e-3	
Conf. model		*	*	-0.030	± 0.003	322	± 31	*	*	1.6e-3	± 1e-5	2.8e-4	± 1e-5	3.00e-4	± 2.88e-5
Conf. model	(39% EO)	*	*	-0.050	± 0.005	420	± 14	*	*	1.1e-3	± 1e-5	8.0e-4	± 1e-5	4.10e-4	± 1.40e-5
Conf. model	(52% EO)	*	*	-0.051	± 0.002	626	± 19	*	*	9.8e-4	± 1e-5	9.9e-4	± 2e-5	6.10e-4	± 1.85e-5
DC-SBM	(11% EO)	165	± 9.0	-0.052	± 0.004	1,403	± 67	1.814	± 0.008	6.7e-4	± 2e-5	1.2e-3	± 4e-5	3.30e-3	± 2.71e-4
ERGM	(56% EO)	243	± 1.94	-0.077	± 0.000	2,293	± 23	1.786	± 0.003	6.9e-4	± 2e-5	1.2e-3	± 1e-5	2.17e-3	± 5.44e-5
BTER	(2% EO)	199	± 13	0.033	± 0.008	3060	± 114	1.787	± 0.004	1.1e-3	± 1e-5	7.5e-4	± 1e-5	4.62e-3	± 5.92e-4
VGAe	(0.3% EO)	13.1	± 1	-0.010	± 0.014	14	± 3	1.674	± 0.001	1.4e-3	± 2e-5	3.2e-4	± 1e-5	1.17e-3	± 2.02e-4
NetGAN VAL	(39% EO)	199	± 6.7	-0.060	± 0.004	1,410	± 30	1.773	± 0.002	6.5e-4	± 1e-5	1.3e-3	± 2e-5	2.33e-3	± 1.75e-4
NetGAN EO	(52% EO)	233	± 3.6	-0.066	± 0.003	1,588	± 59	1.793	± 0.003	6.0e-4	± 1e-5	1.4e-3	± 1e-5	2.44e-3	± 1.91e-4

Graph		Wedge count		Rel. edge distr. entr.		Largest conn. comp		Claw count		Gini coeff.		Edge overlap		Characteristic path length	
		Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.
CITESEER		16,824		0.959		2,110		125,701		0.404		1		10.33	
Conf. model		*	*	0.955	± 0.001	2,011	± 6.8	*	*	*	*	0.008	± 0.001	5.95	± 0.03
Conf. model	(42% EO)	*	*	0.956	± 0.001	2,045	± 12.5	*	*	*	*	0.42	± 0.002	6.14	± 0.03
Conf. model	(76% EO)	*	*	0.957	± 0.001	2,065	± 10.2	*	*	*	*	0.76	± 0.0	6.85	± 0.04
DC-SBM	(6.6% EO)	15,531	± 592	0.938	± 0.001	1,697	± 27	69,818	± 11,969	0.502	± 0.005	0.066	± 0.011	7.75	± 0.26
ERGM	(27% EO)	16,346	± 101	0.945	± 0.001	1,753	± 15	80,510	± 1,337	0.474	± 0.003	0.27	± 0.01	5.92	± 0.01
BTER	(2% EO)	18,193	± 661	0.940	± 0.001	1,708	± 14	113,425	± 19,737	0.491	± 0.007	0.02	± 0.002	5.66	± 0.07
VGAe	(0.2% EO)	8,141	± 47	0.986	± 0.000	2,110	± 0	6,611	± 144	0.256	± 0.003	0.002	± 0.001	7.75	± 0.04
NetGAN VAL	(42% EO)	12,998	± 84.6	0.969	± 0.000	2,079	± 12.6	57,654	± 4,226	0.354	± 0.001	0.42	± 0.006	8.28	± 0.11
NetGAN EO	(76% EO)	15,202	± 378	0.963	± 0.000	2,053	± 23	94,149	± 11,926	0.385	± 0.002	0.76	± 0.01	7.68	± 0.13
CORA-ML		101,872		0.941		2,810		3.1e6		0.482		1		5.61	
Conf. model		*	*	0.928	± 0.002	2,785	± 4.9	*	*	*	*	0.013	± 0.001	4.38	± 0.01
Conf. model	(39% EO)	*	*	0.931	± 0.002	2,793	± 2.0	*	*	*	*	0.39	± 0.0	4.41	± 0.02
Conf. model	(52% EO)	*	*	0.933	± 0.001	2,793	± 6.0	*	*	*	*	0.52	± 0.0	4.46	± 0.02
DC-SBM	(11% EO)	73,921	± 3,436	0.934	± 0.001	2,474	± 18.9	1.2e6	± 170,045	0.523	± 0.003	0.11	± 0.003	5.12	± 0.04
ERGM	(56% EO)	98,615	± 385	0.932	± 0.001	2,489	± 11	3,1e6	± 57,092	0.517	± 0.002	0.56	± 0.014	4.59	± 0.02
BTER	(2% EO)	91,813	± 3,546	0.935	± 0.000	2,439	± 19	2.0e6	± 280,945	0.515	± 0.003	0.02	± 0.001	4.59	± 0.03
VGAe	(0.3% EO)	31,290	± 178	0.990	± 0.000	2,810	± 0	46,586	± 937	0.223	± 0.003	0.003	± 0.001	5.28	± 0.01
NetGAN VAL	(39% EO)	75,724	± 1,401	0.959	± 0.000	2,809	± 1.6	1.8e6	± 141,795	0.398	± 0.002	0.39	± 0.004	5.17	± 0.04
NetGAN EO	(52% EO)	86,763	± 1,096	0.954	± 0.001	2,807	± 1.6	2.6e6	± 103,667	0.42	± 0.003	0.52	± 0.001	5.20	± 0.02

D. Graph statistics during the training process

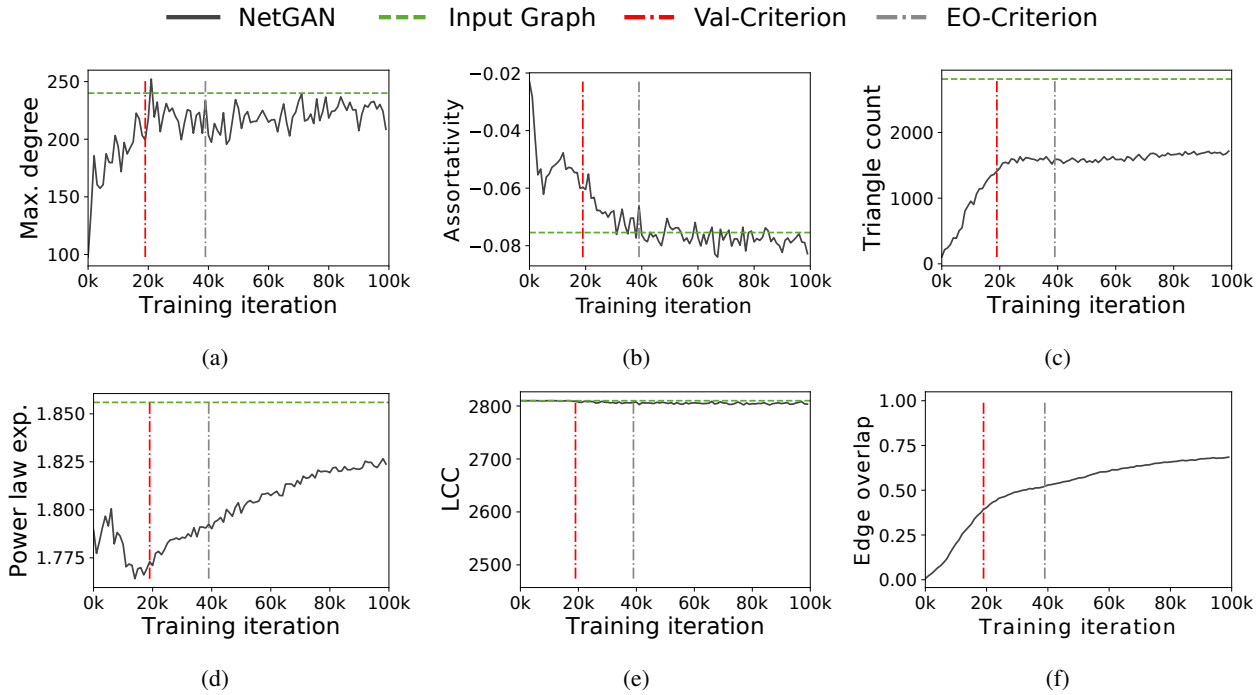


Figure 7: Evolution of graph statistics during training on CORA-ML

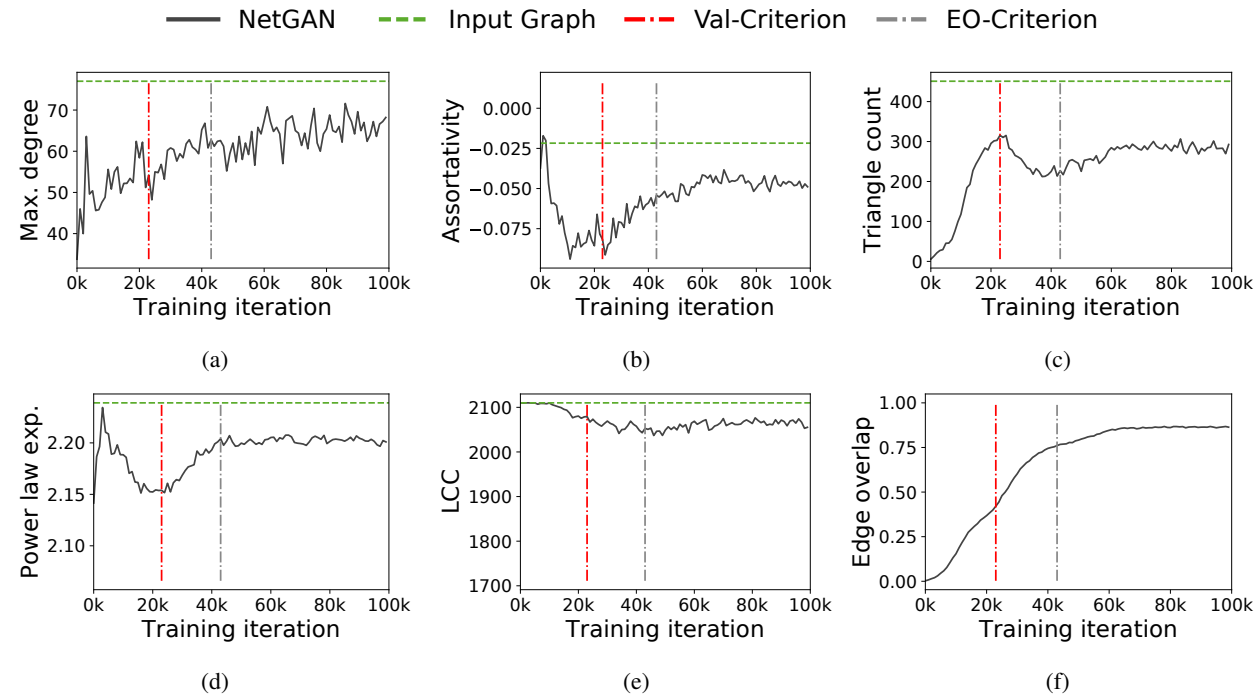
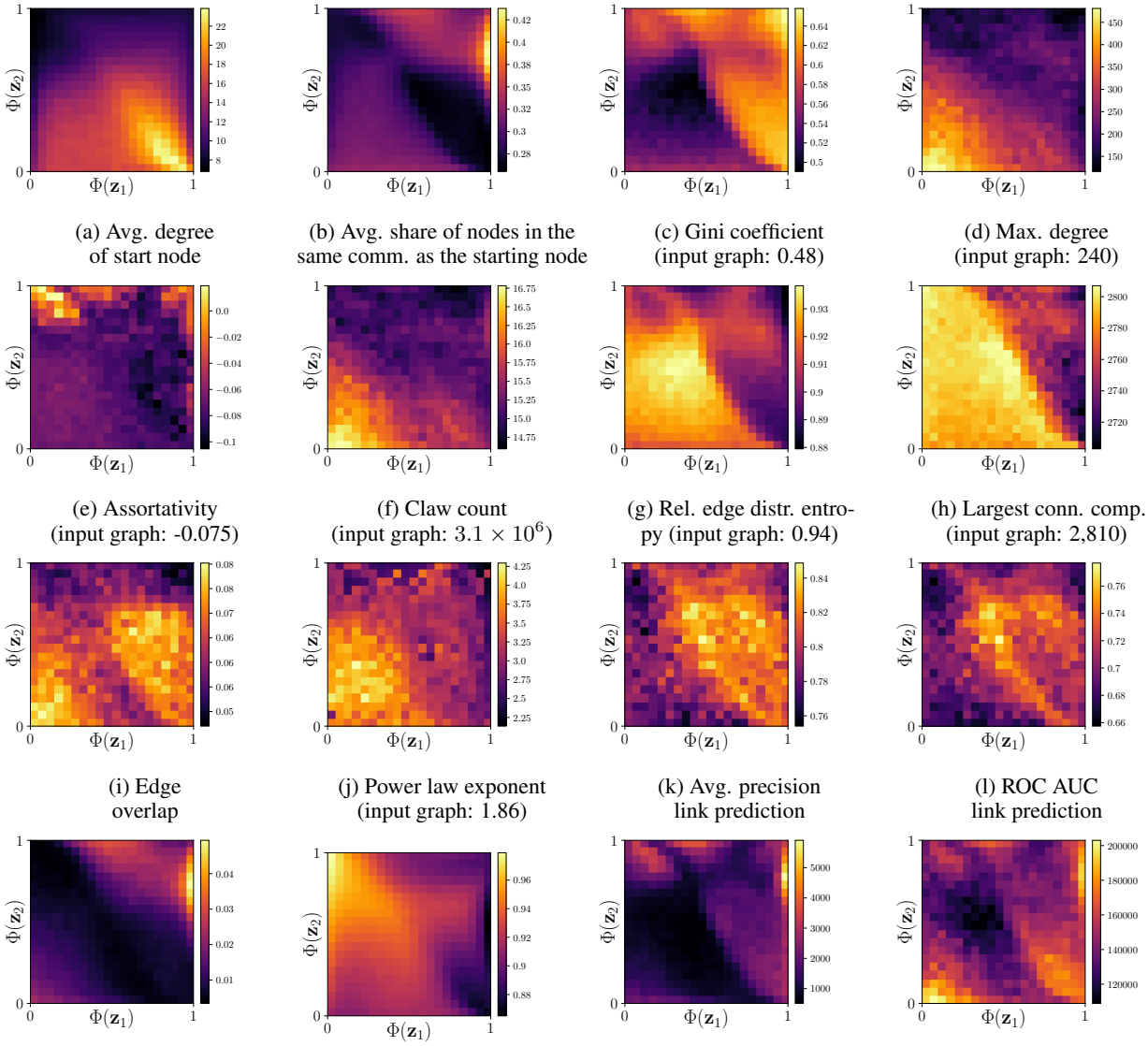


Figure 8: Evolution of graph statistics during training on CITESEER

E. Latent space interpolation heatmaps



(a) Avg. degree of start node

(b) Avg. share of nodes in the same comm. as the starting node

(c) Gini coefficient (input graph: 0.48)

(d) Max. degree (input graph: 240)

(e) Assortativity (input graph: -0.075)

(f) Claw count (input graph: 3.1×10^6)

(g) Rel. edge distr. entropy (input graph: 0.94)

(h) Largest conn. comp. (input graph: 2,810)

(i) Edge overlap

(j) Power law exponent (input graph: 1.86)

(k) Avg. precision link prediction

(l) ROC AUC link prediction

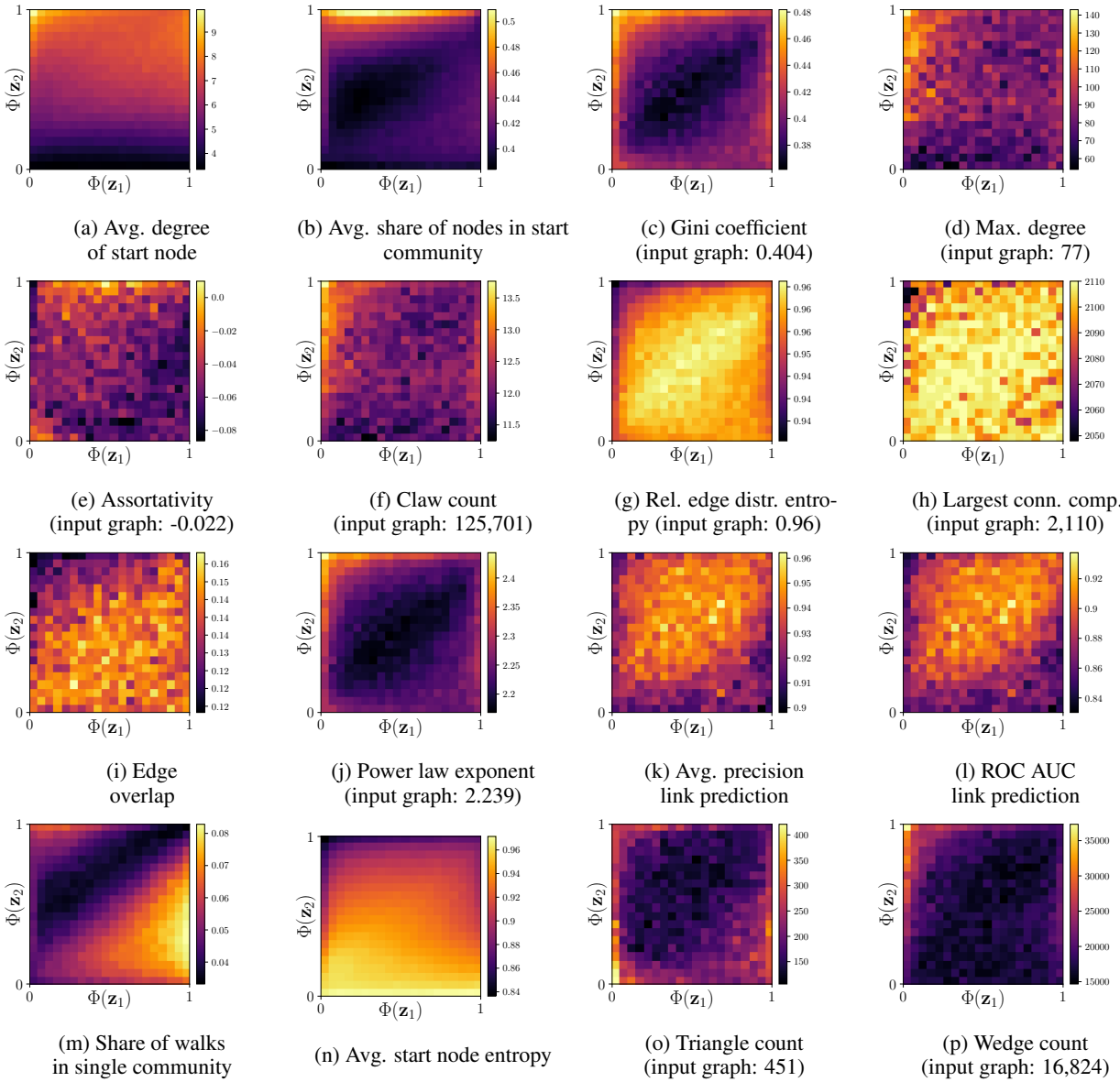
(m) Share of walks in single community

(n) tAvg. start node entropy

(o) Triangle count (input graph: 2,814)

(p) Wedge count (input graph: 101,872)

Figure 9: Properties of the random walks as well as the graphs sampled from the 20×20 latent space bins, trained on CORA-ML.



(a) Avg. degree of start node

(b) Avg. share of nodes in start community

(c) Gini coefficient (input graph: 0.404)

(d) Max. degree (input graph: 77)

(e) Assortativity (input graph: -0.022)

(f) Claw count (input graph: 125,701)

(g) Rel. edge distr. entropy (input graph: 0.96)

(h) Largest conn. comp. (input graph: 2,110)

(i) Edge overlap

(j) Power law exponent (input graph: 2.239)

(k) Avg. precision link prediction

(l) ROC AUC link prediction

(m) Share of walks in single community

(n) Avg. start node entropy

(o) Triangle count (input graph: 451)

(p) Wedge count (input graph: 16,824)

Figure 10: Properties of the random walks as well as the graphs sampled from the 20×20 latent space bins, trained on CITESEER.

F. Latent space interpolation community histograms – CITESEER

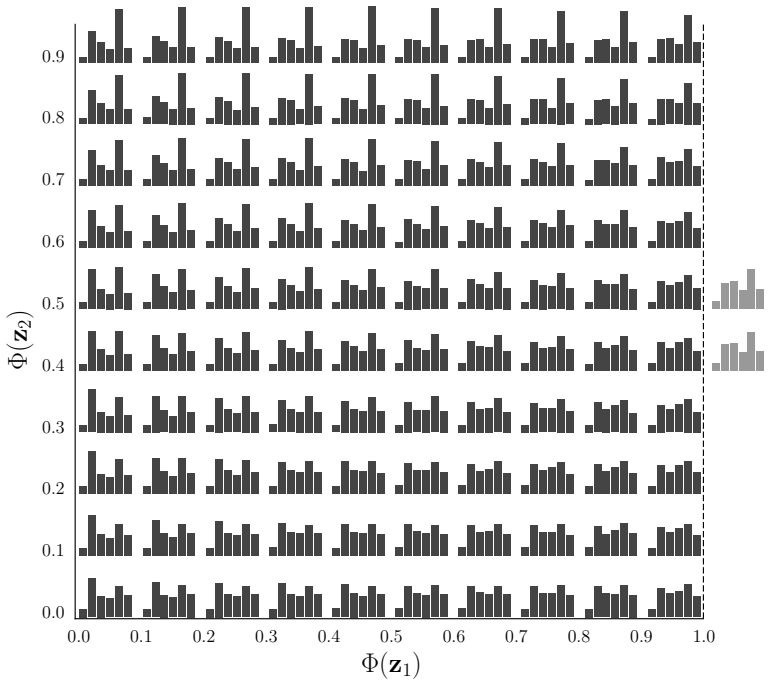


Figure 11: Community distributions of graphs generated by NetGAN on subregions of the latent space $\boldsymbol{z}$, trained on the CITESEER network.

G. Recovering ground-truth edge probabilities

To further investigate the ability of NetGAN to capture the graph structure we perform an additional experiment with the goal of analyzing how well we can recover the ground-truth edge probabilities given a graph generated from a prescribed generative model. Towards that end, first, we generate a graph from DC-SBM ($N = 300$ nodes and 3 communities), then we fit NetGAN on this graph, and finally we compare the ground truth edge probabilities to the edge scores inferred by NetGAN – specifically we compute their ranking correlation. We find a correlation of 0.998 (with EO = 0.42), which shows that NetGAN uncovered the underlying generative process, without overfitting to the input graph.

H. Hyperparameter configuration

As discussed in Sec. 4.2 NetGAN is not sensitive to the choice of most hyperparameters. For completeness, we report here sensible defaults that we used in used in our experiments. The generator and discriminator each have a single hidden layer with 40 and 30 hidden units respectively. The down-projection matrix for the generator is $\boldsymbol{W}_{down,g} \in \mathbb{R}^{N \times H_g}$ with $H_g = 64$, and for the discriminator is $\boldsymbol{W}_{down,d} \in \mathbb{R}^{N \times H_d}$ with $H_d = 32$. The latent code $\boldsymbol{z}$ is drawn from a $d = 16$ dimensional multivariate standard normal distribution. We anneal the temperature from $\tau = 1.0$ down to $\tau = 0.5$ every 500 iterations with a multiplicative decay of 0.995. We tune the parameters p and q (used to bias the generated random walks) for each dataset separately using the procedure in Grover & Leskovec (2016).

We use Adam (Kingma & Ba, 2014) to optimize all the parameters with a learning rate of $1e{-}3$ and we set the regularization strength for the L_2 penalty to $1e{-}6$. We perform five update steps for the parameters of the discriminator for each single update step of the parameters of the generator, and we set the Wasserstein gradient penalty applied to the discriminator to 10 as suggested by Gulrajani et al. (2017). For early stopping, we evaluate the score every 500 iterations, and set the patience to 5 evaluation steps. To calculate the validation score we generate 15M transitions, e.g. for a random walk of length 16 (i.e. 15 transitions per random walk) this equals 1M random walks.

For more details we refer the reader to the provided reference implementation at https://www.kdd.in.tum.de/netgan.