NetGAN without GAN: From Random Walks to Low-Rank Approximations

Luca Rendsburg [1] **Holger Heidrich** [1] **Ulrike von Luxburg** [1 2]

Abstract

A graph generative model takes a graph as input and is supposed to generate new graphs that "look like" the input graph. While most classical models focus on few, hand-selected graph statistics and are too simplistic to reproduce real-world graphs, NetGAN recently emerged as an attractive alternative: by training a GAN to learn the random walk distribution of the input graph, the algorithm is able to reproduce a large number of important network patterns simultaneously, without explicitly specifying any of them. In this paper, we investigate the implicit bias of NetGAN. We find that the root of its generalization properties does not lie in the GAN architecture, but in an inconspicuous low-rank approximation of the logits random walk transition matrix. Step by step we can strip NetGAN of all unnecessary parts, including the GAN, and obtain a highly simplified reformulation that achieves comparable generalization results, but is orders of magnitudes faster and easier to adapt. Being much simpler on the conceptual side, we reveal the implicit inductive bias of the algorithm — an important step towards increasing the interpretability, transparency and acceptance of machine learning systems.

1. Introduction

A graph generative model is a mechanism to achieve the following task: for a given input graph (or a set of input graphs), generate new graphs that have a similar structure as the input graph. The mechanism is supposed to slightly perturb the graph, but should not change its characteristic structure (such as the community structure, the characteristic path lengths, etc). Being able to create perturbed copies of a graph is useful in many different scenarios, for example: comparing a small sample of brain networks for Alzheimer patients (just one, in the extreme case) to a large population of healthy subjects, making robustness statements about a climate network by running a sensitivity analysis on perturbed copies, or performing a generic bootstrap analysis.

A recent graph generative model that has received a lot of attention is NetGAN (Bojchevski et al., 2018). First, it samples a set of random walks from the input graph to train a GAN (Goodfellow et al., 2014), whose generator learns to produce node-sequences that resemble random walks over the input graph. Generated graphs are then obtained as reconstructions based on these sequences. The inherent assumption of this approach is that random walks describe graphs in a reasonably holistic way: local statistics such as motifs are observable in the individual random walks, while global statistics such as cluster structure and diameter are encoded in the distribution over random walk sequences. As opposed to other approaches, NetGAN does not make any explicit model assumptions; rather, it is supposed to implicitly learn many local and global graph statistics simultaneously by reproducing random walk statistics. However, if the goal is to generalize (perturb) the input graph, there has to be an implicit bias as to which type of generalization (perturbation) is preferred (no free lunch). The goal of our paper is to characterize this bias of NetGAN, which we will achieve by reformulating it in terms of a distance function between graphs. This formulation provides insights on the influence of design choices and model parameters, such as the length of the random walks. Scrutinizing the NetGAN architecture, we observe that many of its components can be considerably simplified. Step by step we strip all the unnecessary parts until we are left with the only crucial ingredient, a low-rank approximation of the logits random walk transition matrix. Our main contributions are:

- **Reformulation of NetGAN.** We reformulate NetGAN as a low-rank approximation with respect to the Kullback-Leibler divergence between transition matrices, which requires neither a GAN nor any sampling.
- **Huge speedup.** Our algorithm retains the generalization

[1]Department of Computer Science, University of Tübingen, Germany [2]Max Planck Institute for Intelligent Systems, Tübingen, Germany. Correspondence to: Luca Rendsburg <luca.rendsburg@uni-tuebingen.de>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

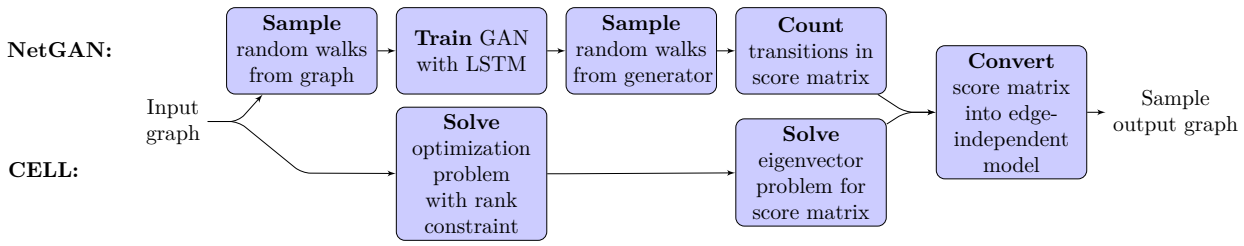

Figure 1. Pipelines for NetGAN (upper path) and our proposed method CELL (lower path). CELL is a condensed version of NetGAN that bypasses the expensive sampling steps and replaces the GAN with an optimization problem.

performance of NetGAN, but runs in seconds instead of hours. See Table 1 for a comparison of training times.

- **Transparency.** Our algorithm is conceptually much simpler than NetGAN. This opens the possibility to analyze it theoretically, and allows for application-specific adaptions.

2. Background: NetGAN

2.1. Graph and random walk notation

Like NetGAN, we consider an unweighted, undirected, and connected graph $G = (V, E)$ with nodes $V = [N] = \{1, \dots, N\}$, edges $E \subseteq V \times V$, and number of edges $e(G) = |E|$. It has adjacency matrix $A \in \{0,1\}^{N\times N}$, degree vector $d \in \mathbb{R}^N$, degree matrix $D = \mathrm{diag}(d) \in \mathbb{R}^{N\times N}$, and the transition matrix for unbiased random walks on G is given by $P = D^{-1}A \in \mathbb{R}^{N\times N}$. We additionally assume G to be non-bipartite so that the random walk described by P has a unique stationary distribution $\pi \in \mathbb{R}^N$. A single random walk of length T is an ordered tuple $R = (v_0, \dots, v_T) \in V^{T+1}$, and a set of n random walks is denoted by $\mathcal{R} = \{R_1, \dots, R_n\}$. The score matrix $S(\mathcal{R}) \in \mathbb{R}^{N\times N}$ counts the transitions in $\mathcal{R}$, that is, $S_{v,w}$ equals the total number of times random walks in $\mathcal{R}$ transition from v to w. If clear from the context, we drop the dependency on $\mathcal{R}$ and write S instead of $S(\mathcal{R})$. An edge-independent random graph model, sometimes also called inhomogeneous Erdős-Rényi model, is a symmetric matrix $A^\dagger \in [0,1]^{N\times N}$ of edge probabilities. Graphs on the same vertices $[N]$ are sampled from this model by drawing $e(G)$ edges $\{v, w\}$ with probability $A^\dagger_{v,w}$ independently and without replacement. We use bold symbols, if we consider an object as a random variable (e. g. $\boldsymbol{R}$ instead of R).

2.2. NetGAN

In this section, we give a high-level overview of the NetGAN algorithm; for more details, we refer the reader to Bojchevski et al. (2018). NetGAN is a graph generative model: given a single input graph G, it returns graphs G' on the same set of nodes by proceeding in two main steps. First, it learns the distribution over random walks drawn from the input graph in the *learning step*. It then reconstructs the graph based on "synthetic" random walks sampled from this learned distribution in the *reconstruction step*. See Figure 1 for a schematic overview.

Table 1. Training time (in seconds) for NetGAN and our proposed method CELL on a variety of networks. NetGAN requires a GPU for training, while CELL runs on a CPU.

DATA SET (NODES/ EDGES)	NETGAN	CELL
CORA-ML (2,810/ 7,981)	7,478	21
CITESEER (2,110/ 3,668)	4,654	10
POLBLOGS (1,222/ 16,779)	55,276	15
RT-GOP (4,687/ 5,529)	14,800	23
WEB-EDU (3,031/ 6,474)	11,000	16

Learning step. Given an input graph G, NetGAN samples a large set $\mathcal{R}$ of random walks of fixed length T with randomly chosen start nodes. These random walks form the training set for a GAN: the generator tries to produce node sequences of length T that resemble the observed random walks in $\mathcal{R}$, while the discriminator tries to distinguish real from generated sequences. Both generator and discriminator use the Long short-term memory architecture (LSTM) (Hochreiter & Schmidhuber, 1997), and they are trained with the Wasserstein loss (Arjovsky et al., 2017). Training finishes once an early stopping criterion is met, after which the generator is used to sample synthetic random walks.

During and after training, the generator constructs each synthetic random walk $(v_0, \dots, v_T)$ in a step-by-step procedure. First, random noise z is used to initialize the memory state m_0 of the LSTM architecture and the start node v_0 of the sequence. A function f_θ with learnable parameters θ then repeatedly updates the two values: given the current memory state m_t and node v_t, it outputs the next memory state m_{t+1} and the distribution p_{t+1} over the next node v_{t+1} in form of logits. The next node v_{t+1} is then obtained as a sample from this distribution. In equations, this update is

described by

$$(m_{t+1}, p_{t+1}) = f_\theta(m_t, v_t)\,, \qquad v_{t+1} \sim \mathrm{Cat}\left(\sigma(p_{t+1})\right)\,, \tag{1}$$

where Cat denotes the categorical distribution and σ the softmax function, which converts the logits into a probability distribution on $[N]$. This procedure is repeated until the sequence has the desired length T.

Reconstruction step. After training is finished, NetGAN uses the generator to generate a large set of n synthetic random walks. Their transitions are counted in a joint score matrix S, which is then converted into an edge-independent random graph model $A^\dagger$ by symmetrizing and then normalizing it, that is,

$$A^\dagger_{k,l} = \frac{\max\{S_{k,l}, S_{l,k}\}}{\sum_{k',l'=1}^{N} \max\{S_{k',l'}, S_{l',k'}\}}\,. \tag{2}$$

To obtain the new graph G', NetGAN samples $e(G)$ edges independently and without replacement from $A^\dagger$ while preventing self-loops and isolated nodes.

3. What causes the generalization?

In this section, we identify those parts of NetGAN that we believe to be absolutely necessary to achieve the two goals of producing new graphs that (i) resemble the input graph by mimicking its graph statistics, but (ii) also generalize the input graph by sharing only a certain amount of its edges. The complicated GAN- and LSTM-based architecture used by NetGAN disguises its underlying bias and makes a direct analysis difficult. Therefore, we examine all the individual steps of NetGAN, not in terms of *how* they work, but *what* they aim to achieve.

The random walks? The intuition of NetGAN is that graphs with a similar random walk distribution also share many of their topological properties. In fact, as we observe in Section 5.3, learning the transition matrix of random walks by counting their transitions is sufficient for perfectly reconstructing the input graph. This excludes the possibility that by reducing graphs to their random walk statistics, we introduce an irreversible systematic bias.

The GAN? The role of the GAN is to learn the random walk distribution of the input graph. We prove in Section 5.3 that if the GAN perfectly learns the random walk distribution, NetGAN will simply reproduce the input graph instead of generalizing it. However, the results reported by Bojchevski et al. (2018) show that even if NetGAN is trained for a long time, it produces graphs that are considerably different from the input graph as measured by edge overlap. Consequently, there has be be another mechanism that prevents the GAN from memorizing the input graph.

The LSTM? As the authors of NetGAN pointed out themselves, the LSTM architecture, which is supposed to capture long-term dependencies, seems to be an odd choice for learning Markov sequences that by construction do not have any such dependencies. It is possible that this architecture choice injects noise into the learning process, which prevents memorization of the input graph. Yet, this type of noise seems to be rather uncontrolled, and we consider it unlikely that this aspect of the LSTM cannot be replaced by a simpler, more direct mechanism.

Computational trick: low-rank approximation. What is left? In our opinion, the only component that explains why NetGAN successfully generalizes graphs is a computational trick: the LSTM is not operating on the high-dimensional space $\mathbb{R}^N$ directly. In order to reduce computational complexity, it uses learnable down- and up-projections $W_{\text{down}} \in \mathbb{R}^{N\times H}$ and $W_{\text{up}} \in \mathbb{R}^{H\times N}$ with $H \ll N$. As we derive in Section A of the supplementary, these projections force the update rule of a node and memory state pair (v_t, m_t) with v_t as one-hot vector to be of the form

$$p_{t+1} = {v_t}^\top W(m_t)\,, \qquad v_{t+1} \sim \mathrm{Cat}\left(\sigma(p_{t+1})\right)\,, \tag{3}$$

where $W(m_t) \in \mathbb{R}^{N\times N}$ depends on m_t and has rank at most H. Because $W(m_t)$ is the transition matrix after applying σ, we refer to it as the *logit transition matrix*. NetGAN forces this matrix to have low rank, which leads us the following conjecture:

Conjecture: The key ingredient of NetGAN is to learn the random walk distribution by performing a low-rank approximation of the logit transition matrix.

To validate this conjecture, we derive a simplified method that applies this low-rank approximation directly and demonstrate its comparable performance in experiments.

4. Stripping NetGAN

We now gradually simplify NetGAN by stripping it of all unnecessary components in Section 4.1. Additionally, we observe in Section 4.2 that sampling random walks can be circumvented with a limit argument. This leads to our new, highly simplified method called Cross-Entropy Low-rank Logits (CELL), see Section 4.3 for a summary and Figure 1 for a schematic outline.

4.1. Low-rank approximation replaces the GAN

Motivated by the above conjecture, we now prune the update rule in Eq. (3) until we arrive at a rank-constrained optimization problem. Justified by the Markov property of unbiased random walks, we first drop the LSTM and the memory state m_t. In Section A of the supplementary, we derive

that the GAN learns the random walk distribution by choosing its transition matrix directly from the parametric family $\mathcal{P} = \{\sigma_{\text{rows}}(W) \in \mathbb{R}^{N\times N} : W \in \mathbb{R}^{N\times N}, \operatorname{rank}(W) \leqslant H\}$, where σ_{rows} denotes the function that applies the softmax σ to each row of a matrix. The training set for this problem consists only of the transitions of random walks in $\mathcal{R}$, and the noise random variable z plays the subordinate role of choosing the first node. This parametric family formulation defeats the purpose of using a GAN at all, which is why instead we revert to the classical maximum likelihood approach (or, equivalently, the cross-entropy loss) on $\mathcal{P}$: using the notation $(k, l) \in \mathcal{R}$ to denote all transitions (with multiple counting) of random walks in $\mathcal{R}$, the resulting problem is given by

$$\begin{aligned} \min_{W\in\mathbb{R}^{N\times N}} & -\sum_{(k,l)\in\mathcal{R}} \log \sigma_{\text{rows}}(W)_{k,l}\,, \\ \text{s.t.} \quad & \operatorname{rank}(W) \leqslant H\,. \end{aligned} \tag{4}$$

In short: instead of learning the random walk distribution by training a GAN, we approximate its transition matrix directly by solving a rank-constrained optimization problem.

4.2. Bypassing random walk sampling

There is another aspect of NetGAN that is somewhat puzzling: even to learn a graph of moderate size, for example the graph CORA-ML with about 3,000 vertices and 8,000 edges, NetGAN needs to sample 7,500,000 random walks of length 15 from the input graph, which are worth 112,500,000 edges. In other words, we see every edge of the input graph about 14,000 times on average — with which any edge-frequency statistic would be very close to its expected value. The same order of magnitude applies to the sampling of random walks from the generator in the reconstruction step. With that observation, a natural question is whether we can circumvent the random walk sampling, and the answer is yes. Since the random walks are only used in form of the score matrix that contains the frequency of node transitions, and this matrix converges for a large number of random walks, we can substitute the actual score matrix with its limit value. The remainder of this section formalizes this idea in Eq. (7) and applies it to NetGAN at both sampling steps.

Convergence of the score matrix S. First, we consider a single random walk $\boldsymbol{R} = (\boldsymbol{v_0}, \ldots, \boldsymbol{v_T})$ of length T as a random variable, whose distribution depends on the distribution $q_0 \in \mathbb{R}^N$ of the first node $\boldsymbol{v_0}$ and the transition matrix P. For $t \in \{1, \ldots, T\}$, let $Q_t \in \mathbb{R}^{N\times N}$ denote the distribution of the t-th transition $(\boldsymbol{v_{t-1}}, \boldsymbol{v_t})$ in $\boldsymbol{R}$. Its marginal $\boldsymbol{v_{t-1}}$ is distributed as $q_{t-1} \in \mathbb{R}^N$ and its conditional $\boldsymbol{v_t}|\boldsymbol{v_{t-1}}$ is distributed as P, which yields the matrix decomposition

$$Q_t = \operatorname{diag}(q_{t-1})P\,. \tag{5}$$

From this perspective, counting the transitions of a single random walk R in a score matrix $S(R) \in \mathbb{R}^{N\times N}$ can be expressed as $S(R) = \sum_{t=1}^{T} \hat{Q}_t(R)$, where $\hat{Q}_t(R)$ is the empirical version of Q_t based on one sample. The score matrix $S = S(R_1, \ldots, R_n)$ based on n random walks $R_1, \ldots, R_n$ decomposes into $S = \sum_{j=1}^{n} S(R_j)$, and with the above considerations we have $S = \sum_{j=1}^{n}\sum_{t=1}^{T} \hat{Q}_t(R_j)$. By the Glivenko-Cantelli theorem for empirical distributions, we can compute the limit of S/n for $n \to \infty$ as

$$\frac{S}{n} = \sum_{t=1}^{T} \frac{1}{n}\sum_{j=1}^{n} \hat{Q}_t(R_j) \xrightarrow[n\to\infty]{\text{a. s.}} \sum_{t=1}^{T} Q_t\,. \tag{6}$$

Using Eq. (5), the normalized right-hand side is given by $\sum_{t=1}^{T} Q_t/T = \operatorname{diag}(\rho_T)P$, where $\rho_T = \sum_{t=1}^{T} q_{t-1}/T$. In that sense, using a large number of random walks reduces to node weights ρ_T. Since the underlying graph is by assumption connected and non-bipartite, the stationary distribution π of P exists and is unique, that is, $\pi = \lim_{t\to\infty} q_t$. Hence the Cesàro mean ρ_T also converges to π as $T \to \infty$. Or, in other words: for any initial distribution q_0, the node weights induced by sufficiently long random walks are given by π. In conjunction with Eq. (6), we obtain the limit of the normalized score matrix

$$\frac{S}{nT} \xrightarrow[n,T\to\infty]{\text{a. s.}} \operatorname{diag}(\pi)P\,. \tag{7}$$

Note that we take two limits to approximate S. We take the first limit with respect to the amount of random walks n, because NetGAN samples many random walks. The second limit with respect to the length T dilutes the influence of the initial distribution ($\rho_1 = q_0$) in favor of the stationary distribution ($\lim_{T\to\infty} \rho_T = \pi$). This is appropriate because most real-world networks have small diameter, and the length $T = 15$ used in NetGAN already ensures that ρ_T is close to its limit distribution. Furthermore, the authors of NetGAN already observed that taking longer random walks increases performance. Finally, in Section 5.4 we will see that the information encoded in the start distribution of the random walk can be more directly incorporated by the node weights.

Replacing random walks from the input graph. The objective in Eq. (4) sums over all node transitions in $\mathcal{R}$. We count the transitions in a corresponding score matrix $S = S(\mathcal{R})$ to rewrite the objective function as

$$-\sum_{k,l=1}^{N} S_{k,l} \log \sigma_{\text{rows}}(W)_{k,l}\,. \tag{8}$$

Normalizing S does not change the minimum, and allows us to approximate it with the limit $\operatorname{diag}(\pi)P$ in Eq. (7). Since we consider unbiased random walks according to $P = D^{-1}A$, the stationary distribution π is proportional to the degrees d, hence $\operatorname{diag}(\pi)P \propto \operatorname{diag}(d)D^{-1}A = A$.

This means that we observe every edge in every direction with the same frequency, see Lovász et al. (1993) for a survey on random walks on graphs. We use this new weighting A to define our final objective function

$$F(W) = -\sum_{k,l=1}^{N} A_{k,l} \log \sigma_{\text{rows}}(W)_{k,l} \tag{9}$$

and our final objective

$$\begin{aligned} &\min_{W \in \mathbb{R}^{N \times N}} F(W)\,, \\ &\text{s.t.} \quad \operatorname{rank}(W) \leqslant H\,, \end{aligned} \tag{10}$$

whose solution is denoted as W^*. Note that the sum in Eq. (9) grows only as $\mathcal{O}(e(G))$, and we can enforce the rank-constraint in Eq. (10) with the factorization $W = W_{\text{down}} W_{\text{up}}$, where $W_{\text{down}} \in \mathbb{R}^{N \times H}, W_{\text{up}} \in \mathbb{R}^{H \times N}$, resulting in $\mathcal{O}(NH)$ trainable parameters and a non-convex optimization problem.

Replacing random walks from the generator. In principle, we could use the synthetic transition matrix $P^* = \sigma_{\text{rows}}(W^*)$ defined with the solution W^* of Eq. (10) in place of the generator: we produce synthetic random walks of length T, with transition matrix P^*, and with the same distribution over the first node as in the training set, and then count their transitions in a score matrix. But since the score matrix is needed only up to proportionality for the edge-independent model, we can use the limit in Eq. (7) instead, which replaces sampling random walks with solving the eigenvector problem $\pi^{*\top} P^* = \pi^{*\top}$. That is, we skip sampling random walks and simply set $S = \operatorname{diag}(\pi^*) P^*$.

4.3. Our proposed algorithm: Cross-Entropy Low-rank Logits (CELL)

In the previous section, we have shown how to (i) replace the LSTM and GAN architecture with a low-rank approximation of the logit transition matrix with respect to the cross-entropy loss, (ii) replace sampling random walks from the input graph with using its adjacency matrix directly, and (iii) replace sampling random walks from the generator with solving an eigenvector problem. The result of this analysis is our simplified algorithm Cross-Entropy Low-rank Logits (CELL), summarized in Algorithm 1. It takes the adjacency matrix A of a graph G as input and returns a symmetric matrix $A^\dagger$ of edge probabilities, from which new graphs G' can be sampled. For solving optimization problem (10), we factorize $W = W_{\text{down}} W_{\text{up}}$ with $W_{\text{down}} \in \mathbb{R}^{N \times H}$ and $W_{\text{up}} \in \mathbb{R}^{H \times N}$ to satisfy the rank constraint, and optimize with Adam (Kingma & Ba, 2014). Training continues until a stopping criterion is met, for which we pause at regular intervals and generate new graphs to evaluate the stopping criterion. In this paper, we consider the criterion of reaching a predefined edge overlap of generated graphs and input graph, see Section 6.1.

Algorithm 1 Cross-Entropy Low-rank Logits (CELL)[1]

input adjacency matrix $A \in \{0,1\}^{N \times N}$, rank $H \ll N$
output matrix of edge probabilities $A^\dagger \in [0,1]^{N \times N}$

1: Solve optimization problem (10) for W^*
2: Compute transition matrix: $P^* \leftarrow \sigma_{\text{rows}}(W^*)$
3: Solve eigenvalue problem $\pi^{*\top} P^* = \pi^{*\top}$ for π^*
4: Compute score matrix: $S \leftarrow \operatorname{diag}(\pi^*) P^*$
5: Convert score matrix S to edge-independent model $A^\dagger$: $S^\dagger \leftarrow \max\{S, S^\top\}$; $A^\dagger \leftarrow S^\dagger / \text{sum}(S^\dagger)$

return $A^\dagger$

5. Conceptual analysis

Our simple reformulation of NetGAN now opens the possibility to formally analyze the inductive bias associated with its components and allows for user-specific adaptations.

5.1. Inductive bias of NetGAN

Our analysis has shown that the graphs produced by NetGAN come from the class of graphs whose logit transition matrix has a low rank. Note that this does *not* imply that the transition matrix itself has low rank. Even if W^* is trained to have low rank, the corresponding synthetic transition matrix $P^* = \sigma_{\text{rows}}(W^*)$ can have full rank, as is visualized by the eigenvalues in Figure 2. Additionally, our experiments in Section 6.2 suggest that approximating the transition matrix with a low rank matrix and the Frobenius norm as loss function does not achieve good generalization performance. However, minimizing the cross-entropy loss for approximation instead yields generalization performance comparable to the one of NetGAN and CELL. Since the cross-entropy corresponds to the Kullback-Leibler (KL) divergence (see Section B of the supplementary), this suggests that **using the KL divergence as distance measure for approximating transition matrices is the reason for the good generalization performance. On a high level, NetGAN generalizes a graph by choosing new graphs, whose transition matrix is similar in terms of KL-divergence, from a restricted set of graphs.** Whether this restriction is realized by a low-rank assumption on the logits or on the transition matrices itself is not essential, although the former is computationally more feasible.

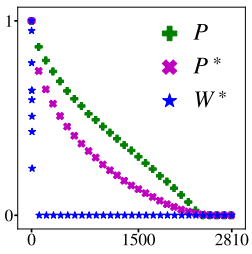

Figure 2. Portion of absolute eigenvalues (sorted and rescaled) for CORA-ML with CELL trained to 50% edge overlap for $H = 9$.

[1] Code available at https://github.com/hheidrich/CELL

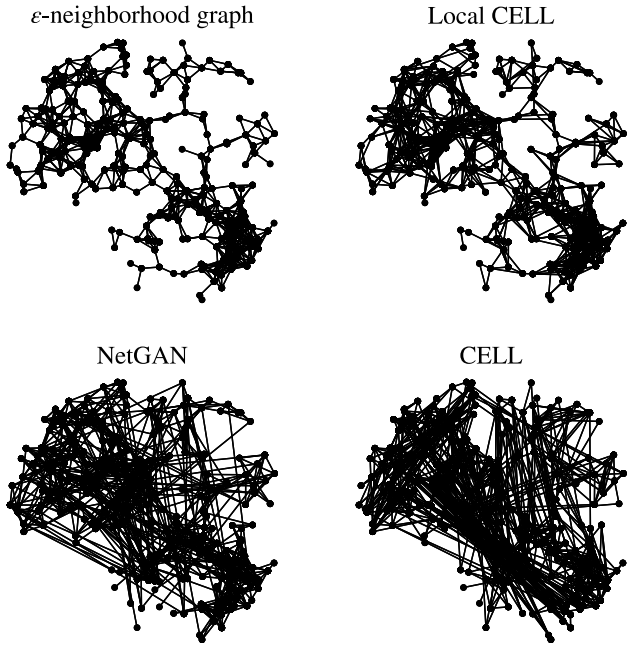

Figure 3. Comparison of an ε-neighborhood graph (top left) with graphs generated by Local CELL, a version of our method biased towards short edges, NetGAN, and our method CELL. Only Local CELL does not generate edges between distant points.

5.2. Bias of the optimization objective and resulting hard examples

We optimize the objective function in Eq. (9) to learn the random walk distribution in form of its transition matrix. By inspecting the objective, we can understand how this is achieved: the synthetic transition matrix $\sigma_{\text{rows}}(W)$ is rewarded *directly* for putting mass on edges of the input graph ($A_{k,l} = 1$). But because the total mass is limited ($\sum_l \sigma_{\text{rows}}(W)_{k,l} = 1$), it is only penalized *indirectly* for wasting mass on non-edges ($A_{k,l} = 0$). In particular, there is no distinction between different non-edges. This hints towards **poor performance for graphs with strong restrictions on the set of edges we deem realistic**, because there is no notion of "bad" edges that could prevent their generation; a possible remedy to this problem is extending the objective function with such a notion.

We illustrate this effect with the example of ε-neighborhood graphs in Figure 3. Here, we want to avoid the generation of edges between nodes with large distance in the Euclidean space, which is not taken into account by NetGAN and CELL. However, a simple adaptation of our method, denoted as "Local CELL", can prevent long edges without loss of generalization performance. The corresponding experiment is provided in Section C.4 of the supplementary.

5.3. No bias in the reconstruction step

A natural question is whether our method might be able to generalize even without the rank constraint in the learning step. Or, phrased differently, whether the reconstruction step introduces a generalization bias. This is not the case. We derive in Section B of the supplementary that without any rank constraint, we exactly recover the input transition matrix as $P^* = P$. Because of $\operatorname{diag}(\pi)P \propto A$, this also holds for the score matrix

$$S = \operatorname{diag}(\pi^*)P^* \propto A\,. \tag{11}$$

Since A is already symmetric, the edge-independent model is given by $A^\dagger \propto A$. This model is equivalent to uniformly sampling edges from the input graph G, and sampling $e(G)$ edges from this model without replacement means sampling all of them. Hence it simply returns the input graph with zero variance. Therefore, **reconstructing the graph with an edge-independent model does not contribute to generalization.** Another interpretation of this observation is that random walks are sufficient to learn a graph in principle.

5.4. Influence of the random walk parameters

For NetGAN it is still unclear how the length T and the start distribution for the first node q_0 of the random walks influence the generated graphs. We derived in Section 4.2 that it does not exploit any complicated patterns in the random walk paths, but simply counts the transitions, which comes down to a weighting of the nodes. When translating NetGAN to our approach, we observe that **the random walk length controls how much influence the start distribution has on the node weights:** instead of taking the limit $T \to \infty$ in the derivation of Section 4.2, we could have completed the analysis with the node weights $\rho = \sum_{t=1}^{T} q_{t-1}/T$ to arrive at the parametrized objective function

$$F_\rho(W) = -\sum_{k,l=1}^{N} \frac{\rho_k}{d_k} A_{k,l} \log \sigma_{\text{rows}}(W)_{k,l}\,. \tag{12}$$

This allows for further interpretation and adaption:

Random walks of length one are sufficient. The random walk parameters T and q_0 are relevant for Eq. (12) only because they determine the node weights ρ. On the other hand, all possible node weights ω can be realized by choosing $q_0 = \omega$ and $T = 1$. This implies for NetGAN that **only using random walks of length one imposes no restriction**, if the distribution of the start node is considered as a hyperparameter instead.

An example that is now readily explained is the setting in Jalilifard et al. (2019). They observe empirically that using short random walks in NetGAN reduces the performance, and propose to counteract by choosing the start

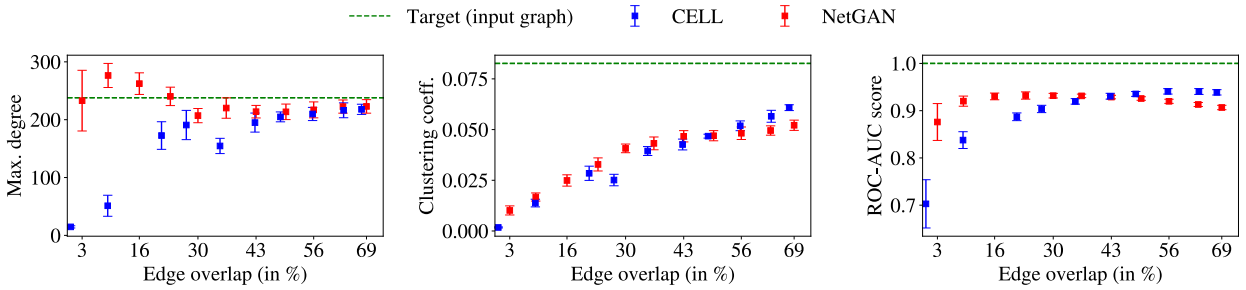

Figure 4. Mean and standard deviation for five trials on CORA-ML, plotted against edge overlap with the input graph. Aside from different initializations, NetGAN and CELL display similar behavior. Additional experiments are provided in the supplementary, Section C.6.

distribution as the density function described in Zhou et al. (2009). Within our framework, this is explained by the node weights: for short random walks, they are close to the uniform distribution (the start distribution of NetGAN), which overemphasizes nodes with low degree and results in bad performance. Choosing the start distribution closer to the stationary distribution instead has the same effect on the node weights as using long random walks.

Node weights ρ as a hyperparameter. Instead of indirectly setting the node weights ρ through the random walk parameters q_0 and T as is done in NetGAN, we can treat ρ as a hyperparameter directly to incorporate beliefs about the graph. Weighting nodes according to the stationary distribution assigns equal weight to all *edges* in Eq. (12). In general, increasing the weight of a node encourages generated graphs to include its adjacent edges. This enables us, for example, to "protect" a certain set of nodes in the sense of preserving their neighborhoods in the generated graphs by increasing their weight.

6. Experiments

The purpose of this section is to (i) verify that CELL has performance comparable to NetGAN while being much faster, and (ii) demonstrate the importance of the cross-entropy loss and benefit of the logit-transformation by comparing with other low-rank approximation baselines.

6.1. Setup of the experiments

Data sets and preprocessing. We experiment on a variety of graph data sets: the citation networks CORA-ML (McCallum et al., 2000) and CITESEER (Sen et al., 2008), the political blogs network POLBLOGS (Adamic & Glance, 2005), the retweet network RT-GOP, and the web graph WEB-EDU (Gleich et al., 2004). All graphs except for CORA-ML are taken from Rossi & Ahmed (2015). For CORA-ML, we use the same preprocessed version as Bojchevski et al. (2018), an overview of the data sets is given in Table 3. We preprocess the graphs by removing loops, edge weights, and edge directions. We then restrict them to their largest connected component to ensure that they are connected. For evaluating the link prediction performance during and after training, we split each graph into training-, validation-, and test-set by taking out 10% of the edges for validaton and another 5% for testing, while ensuring that the remaining graph stays connected. The validation set is only used for the VAL-criterion, an alternative stopping criterion based on link prediction performance that is described in Section C.3 of the supplementary.

Baselines. We compare our model CELL to NetGAN (Bojchevski et al., 2018) and a number of non-parametric baselines: the configuration model, which simply rewires some randomly chosen edges (Molloy & Reed, 1995), and low-rank approximations of the adjacency matrix (LR-Adj), the random walk transition matrix (LR-Trans), the symmetric normalized Laplacian (LR-Lap), and the modularity matrix (LR-Mod), in a similar framework as described by Baldesi et al. (2018). To investigate the contribution of the logit transformation for CELL, we additionally consider a low-rank approximation of the transition matrix with respect to the cross-entropy loss instead of using the Frobenius norm (LR-CE). The original paper by Bojchevski et al. (2018) also compared to a number of parametric baselines, which have the purpose of explicitly fitting some hand-selected graph parameters, but fail to reproduce others. For brevity we do not report the results of these parametric baselines.

Setup and evaluation metrics. To make the results comparable, we train CELL and NetGAN until the same stopping criterion of 52% edge overlap with the input graph is satisfied. This is done by pausing the training at regular intervals, generating a single graph, and calculating the ratio of shared edges to input edges. While NetGAN is trained on a GPU, only a CPU is required for training CELL.

Our first evaluation metric is a set of common graph statistics for input and generated graphs, whose purpose is to measure the extent to which the newly generated graphs

Table 2. Graph statistics and link prediction performance on CORA-ML for generated graphs from NetGAN, our method CELL, and baselines, averaged over five trials. Statistics that are matched by model design for the configuration model are indicated as $*$, and cases that are not applicable as $-$. CELL produces statistics comparable to NetGAN, but is orders of magnitudes faster. This experiment is repeated for all other data sets in Section C.5 of the supplementary.

GRAPH	MAX. DEGREE	ASSORT-ATIVITY	TRIANGLE COUNT	SQUARE COUNT	POWER LAW EXP.	CLUSTER-ING COEFF.	CHARAC. PATH LEN.	ROC-AUC SCORE	TIME (IN S)
CORA-ML	238	-0.076	2,802	14,268	1.86	8.26e-2	5.63	1	–
CONF. MODEL (52% EO)	$*$	-0.053	623	3111	$*$	1.96e-2	4.43	–	1
LR-ADJ (53% EO)	121	-0.042	444	1,128	1.72	2.78e-2	5.17	0.561	32
LR-TRANS (57% EO)	139	-0.058	558	1,617	1.77	2.94e-2	5.07	0.709	33
LR-LAP (52% EO)	167	-0.084	691	1942	1.79	2.79e-2	4.76	0.800	38
LR-MOD (53% EO)	122	-0.043	437	1,135	1.72	2.75e-2	5.17	0.557	48
LR-CE (52% EO)	193	-0.068	1,388	6,284	1.79	5.68e-2	5.37	0.950	73
NETGAN (54% EO)	219	-0.071	1,461	5,555	1.80	5.23e-2	5.13	0.950	7,478
CELL (53% EO)	204	-0.070	1,396	6,880	1.82	5.07e-2	5.26	0.938	21

Table 3. Data sets used. Nodes and edges refer to the largest connected component.

NAME	NODES	EDGES
CORA-ML	2,810	7,981
CITESEER	2,110	3,668
POLBLOGS	1,222	16,779
RT-GOP	4,687	5,529
WEB-EDU	3,031	6,474

reproduce network patterns of the input graph. Since memorizing the input graph trivially reproduces all of its graph statistics, we additionally evaluate the generalization properties in a link prediction task. To do so, we use the edges in the test set and an equal amount of randomly chosen non-edges from the original graph. After training, these are presented to the generative models, which try to classify them as existent or non-existent in the original graph on the basis of the score matrix (or, equivalently, the edge-independent model $A^{\dagger}$). This matrix is produced by all considered models except for the configuration model. A high value in the score matrix suggests the existence of the corresponding edge, while a low value suggests that the edge did not exist in the original graph. The performance is measured by the ROC-AUC score (Area Under Curve for Receiver Operating Characteristic curve), applied to the score matrix evaluated at the edges in question.

6.2. Evaluation

CELL vs. NetGAN. The results for graphs generated on CORA-ML are presented in Table 2. Compared to the other baselines, **CELL generates graphs with statistics close to those of NetGAN** and has similar link prediction performance; some of their small differences might be attributed to the noise of the LSTM used by NetGAN, and to the different optimization procedures. The latter can be observed in Figure 4, which shows the evolution of generated graph statistics during training: NetGAN starts off with a different initialization, but as training continues, the generated graph statistics get close to the target well before memorizing the input graph. Further confirmation of this behavior is given in Sections C.6 and C.7 of the supplementary. However, the most striking difference is the training time, for which our method is **orders of magnitudes faster**, see Table 1.

CELL vs. baselines. Almost all **baselines fail to reproduce most of the graph statistics**, while CELL is reasonably close to all of them. Only LR-CE, the version of our method without the logit space, has performance very similar to CELL. This hints towards the **importance of the cross-entropy loss** rather than the logit space for successfully generalizing a graph. However, using the logit space still has the advantage of requiring only a small rank ($H = 9$ for CELL as compared to $H = 950$ for LR-CE), which results in less trainable parameters and shorter training time.

7. Discussion and future work

We derived a condensed version of NetGAN by identifying its essential steps and performing them directly. We verified experimentally that it retains the generalization performance of NetGAN, but is much faster. Additionally, our simple formulation of the algorithm makes it more accessible for analysis and application-specific extensions.

Analysis. In essence, we revealed the initial random-walk-based approach to be a low-rank approximation of the random walk transition matrix in the logit space. More naive low-rank approximations of matrices related to the input graph do not achieve competitive performance when ap-

proximating with respect to the Frobenius norm, but do so for the cross-entropy loss — a curious fact that we plan to investigate in future work. Based on our new, simplified methods we could analyze the inductive biases of the different components and the role of the parameters of NetGAN. For example, we discover that length and choice of start node of the random walks amount to a weighting of the nodes, which controls their importance in the graph generation process. Based on our better understanding of the bias, we can construct examples which both NetGAN and our algorithm cannot treat in a satisfactory manner.

Extensions. We demonstrated that our method is easily extendable by manipulating the loss function. An additional loss term can prevent the generation of edges we deem undesired, and node weights can emphasize user-specified nodes. Because learning step and reconstruction step are independent, each of them could be replaced by a different procedure. For example, instead of sampling from an edge-independent model, a more general method would sample independent paths to further emphasize locality.

Conclusion. Beyond the particular case of NetGAN, our work is part of a more high-level agenda. Machine learning is used in diverse applications, often not by machine learning experts, and the outcome of algorithms might have considerable impact in science and society. In such a context it is particularly important that our community actively attempts to understand the inherent inductive biases, strengths, and also the weaknesses of algorithms. Finding examples where an algorithm works is important — but maybe even more important is to understand under which circumstances the algorithm produces misleading results. For graph generative models, this might concern medical studies on brain graphs or geoscience studies on climate graphs. We should work hard to make our algorithms as transparent and interpretable as possible. This paper is a small step in that direction.

Acknowledgements

This work has been supported by the German Research Foundation through the Cluster of Excellence "Machine Learning – New Perspectives for Science" (EXC 2064/1 number 390727645), the BMBF Tübingen AI Center (FKZ: 01IS18039A), and the International Max Planck Research School for Intelligent Systems (IMPRS-IS).

References

Adamic, L. A. and Glance, N. The political blogosphere and the 2004 US election: divided they blog. In *Proceedings of the 3rd international workshop on Link discovery*, 2005.

Arjovsky, M., Chintala, S., and Bottou, L. Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

Baldesi, L., Butts, C. T., and Markopoulou, A. Spectral graph forge: Graph generation targeting modularity. In *Conference on Computer Communications*, 2018.

Boguná, M., Pastor-Satorras, R., Díaz-Guilera, A., and Arenas, A. Models of social networks based on social distance attachment. In *Physical review E*, 2004.

Bojchevski, A., Shchur, O., Zügner, D., and Günnemann, S. NetGAN: Generating graphs via random walks. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

Gleich, D. F., Zhukov, L., and Berkhin, P. Fast parallel PageRank: A linear system approach. Technical report, Yahoo! Research Labs, 2004.

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 27*, 2014.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. In *Neural computation*, 1997.

Jalilifard, A., Caridá, V., Mansano, A., and Cristo, R. Can netgan be improved by short random walks originated from dense vertices? *arXiv preprint arXiv:1905.05298*, 2019.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2014.

Lovász, L. et al. Random walks on graphs: A survey. In *Combinatorics, Paul erdos is eighty*, 1993.

McCallum, A. K., Nigam, K., Rennie, J., and Seymore, K. Automating the construction of internet portals with machine learning. In *Information Retrieval*, 2000.

Molloy, M. and Reed, B. A critical point for random graphs with a given degree sequence. In *Random Structures & Algorithms*, 1995.

Orlitsky, A. Estimating and computing density based distance metrics. In *Proceedings of the 22nd international conference on Machine learning*, 2005.

Rossi, R. A. and Ahmed, N. K. The network data repository with interactive graph analytics and visualization. In *Association for the Advancement of Artificial Intelligence*, 2015.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. Collective classification in network data. In *AI magazine*, 2008.

Tenenbaum, J. B., De Silva, V., and Langford, J. C. A global geometric framework for nonlinear dimensionality reduction. In *science*, 2000.

Zhou, Y., Cheng, H., and Yu, J. X. Graph clustering based on structural/attribute similarities. In *Proceedings of the VLDB Endowment*, 2009.

NetGAN without GAN: From Random Walks to Low-Rank Approximations Supplementary Material

Luca Rendsburg **Holger Heidrich** **Ulrike von Luxburg**

A. Replacing the GAN

In this section, we derive our objective in Eq. (4) by inspecting the high-level structure of NetGAN. The learnable projection matrices are given by $W_{\text{down}} \in \mathbb{R}^{N \times H}$ and $W_{\text{up}} \in \mathbb{R}^{H \times N}$ with $H \ll N$. Given the current node v_t as a one-hot vector and suppressing the next memory state m_{t+1} in notation, the generator f_θ can be written as

$$p_{t+1} = f_\theta\left(m_t, v_t\right) = g_\theta\left(m_t, {v_t}^\top W_{\text{down}}\right) W_{\text{up}}\,, \tag{13}$$

where $g_\theta \colon \mathbb{R}^H \to \mathbb{R}^H$ is the part of f_θ that operates on the low-dimensional space. We collect the row vectors $g_\theta\left(m_t, {v_t}^\top W_{\text{down}}\right)$ in a matrix $\widetilde{W}_{\text{down}}(m_t) \in \mathbb{R}^{N \times H}$ and define the product $W(m_t) \coloneqq \widetilde{W}_{\text{down}}(m_t) W_{\text{up}} \in \mathbb{R}^{N \times N}$ to obtain

$$p_{t+1} = {v_t}^\top \widetilde{W}_{\text{down}}(m_t) W_{\text{up}} = {v_t}^\top W(m_t)\,. \tag{14}$$

Therefore, $W(m_t)$ simply serves as logit transition matrix for the random walks. Because of the factorization that defines $W(m_t)$, its rank is at most H.

To derive how exactly we can replace the GAN with a low-rank approximation, we first simplify the update rule in Eq. (14) by dropping the LSTM and with it the dependency on the memory state m_t; this is justified by the Markov property of unbiased random walks. What remains is a matrix W, whose learnable parameters are intertwined with the low-dimensional part g_θ of the generator:

$$p_{t+1} = {v_t}^\top W = g_\theta\left({v_t}^\top W_{\text{down}}\right) W_{\text{up}}\,. \tag{15}$$

Motivated by the assumption that the identity function $\mathrm{Id} \colon \mathbb{R}^H \to \mathbb{R}^H$ can be represented as g_θ, we drop the structural restriction imposed by g_θ, leaving us with $W = W_{\text{down}} W_{\text{up}}$ and update rule

$$p_{t+1} = {v_t}^\top W_{\text{down}} W_{\text{up}}\,. \tag{16}$$

The new update of node v_t is thereby realized by sampling from the categorical distribution of the corresponding row $\sigma(W_{v_t})$, that is, $v_{t+1} \sim \mathrm{Cat}(\sigma(W_{v_t}))$. In this form, training the GAN is equivalent to learning the random walk transition matrix directly from the parametric family $\mathcal{P} = \{\sigma_{\text{rows}}(W) \in \mathbb{R}^{N \times N} \,:\, W \in \mathbb{R}^{N \times N},\, \mathrm{rank}(W) \leqslant H\}$, where σ_{rows} denotes the function that applies σ to each row of a matrix. We then proceed as described in the main paper by learning the transition matrix from this parametric family directly with the maximum likelihood approach.

B. Information-theoretic representation of objective function F

When considering distributions in this section, we let any matrix with positive entries refer to the uniquely determined distribution that is obtained after normalization. We can reformulate our objective F, defined in Eq. (9), in terms of information-theoretic quantities to determine its minimum irrespective of the rank constraint. To do so, we consider node transitions as a random variable $(\boldsymbol{v}, \boldsymbol{w})$ on $[N] \times [N]$. As derived for Eq. (9) in case of node transitions on the input graph, $(\boldsymbol{v}, \boldsymbol{w})$ is distributed according to the adjacency matrix A, wherefore the corresponding conditional distribution of $\boldsymbol{w}|\boldsymbol{v}$ is given by P. The synthetic transition matrix $\sigma_{\text{rows}}(W)$ represents another conditional distribution for $\boldsymbol{w}|\boldsymbol{v}$. From this

perspective, we can reformulate F as

$$\begin{aligned} F(W) = -\sum_{v,w=1}^{N} A_{v,w} \log \sigma_{\text{rows}}(W)_{v,w} &\propto -\mathbb{E}_{(v,w)\sim A}\left[\log \sigma_{\text{rows}}(W)_{v,w}\right] \\ &= -\mathbb{E}_{(v,w)\sim A}\left[\log A_{v,w}\right] + \mathbb{E}_{(v,w)\sim A}\left[\log\left(\frac{A_{v,w}}{\sigma_{\text{rows}}(W)_{v,w}}\right)\right] \\ &= \mathrm{H}_A(\boldsymbol{w}|\boldsymbol{v}) + \mathrm{KL}\left(A(\boldsymbol{w}|\boldsymbol{v}) \,||\, \sigma_{\text{rows}}(W)(\boldsymbol{w}|\boldsymbol{v})\right) . \end{aligned}$$

The first term on the right-hand side is the conditional entropy of the true underlying node transition distribution A and does not depend on W. The second is the conditional relative entropy between the true node transition distribution A, whose conditional is given by P, and the learned conditional $\sigma_{\text{rows}}(W)$. This shows that F is minimized by any W satisfying $\sigma_{\text{rows}}(W) = P$, which makes the low-rank constraint necessary for generalization in the learning step.

C. Experiments

C.1. Graph statistics

Definition of various graph statistics used in this paper. Part of the table is extracted from Bojchevski et al. (2018).

Table 4. Graph statistics for a graph $G = (V, E)$ with $N = |V|$ nodes and $m = |E|$ edges.

GRAPH STATISTIC	COMPUTATION	DESCRIPTION
ASSORTATIVITY	$\frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y}$	PEARSON CORRELATION OF DEGREES OF CONNECTED NODES, WHERE THE (x_i, y_i) PAIRS ARE THE DEGREES OF CONNECTED NODES.
POWER LAW EXPONENT	$1 + n\left(\sum_{v\in V} \log \frac{d(v)}{d_{\text{min}}}\right)^{-1}$	EXPONENT OF THE POWER LAW DISTRIBUTION, WHERE d_{MIN} DENOTES THE MINIMUM DEGREE IN A NETWORK.
RELATIVE EDGE DISTRIBUTION ENTROPY	$-\frac{1}{\log N}\sum_{v\in V} \frac{d(v)}{2m} \log \frac{d(v)}{2m}$	NORMALIZED ENTROPY OF THE DEGREE DISTRIBUTION, 1 MEANS UNIFORM, 0 MEANS A SINGLE NODE IS CONNECTED TO ALL OTHERS.
GINI COEFFICIENT	$\frac{2\sum_{i=1}^{N} i\hat{d}_i}{N\sum_{i=1}^{N}\hat{d}_i} - \frac{N+1}{N}$	COMMON MEASURE FOR INEQUALITY IN A DISTRIBUTION, WHERE $\hat{d}$ IS THE SORTED LIST OF DEGREES IN THE GRAPH.
CHARACTERISTIC PATH LENGTH	$\frac{1}{N(N-1)}\sum_{u\neq v} d(u,v)$	AVERAGE SHORTEST PATH LENGTH, WHERE $d(u,v)$ IS THE SHORTEST PATH LENGTH BETWEEN NODES u AND v.
SPECTRAL GAP	$\lambda_1(L)$	SMALLEST NON-ZERO EIGENVALUE λ_1 OF THE GRAPH LAPLACIAN $L = D - A$.
MOTIF COUNT	–	NUMBER OF COPIES OF H CONTAINED IN G AS A SUBGRAPH. CONSIDERED MOTIFS ARE WEDGES, TRIANGLES, AND SQUARES.

C.2. Baselines

- **Configuration model.** We randomly sample a fraction of the edges in the input graph (fraction stated in brackets), and then rewire the remaining edges by severing them and randomly matching the stubs. This yields a graph with the same degree distribution as in the input graph. Because the resulting graph is not simple in general, we then remove all loops and multiple edges (with high probability, there are only few of them).
- **Low-rank approximations with respect to Frobenius norm.** A class of graph generative models similar in spirit to our method is the Spectral Graph Forge framework, which is based on performing low-rank approximations of matrices derived from the input adjacency matrix A. The pipeline consists of the following steps:

1. Transform A into any derived matrix $M = M(A)$.
2. Perform a low-rank approximation of M to obtain $\tilde{M}$.
3. Back-transform $\tilde{M}$ to $\tilde{A}$ by applying the inverse of the transformation.
4. Obtain edge-independent model $A^\dagger$ by making $\tilde{A}$ symmetric and then normalizing it.
5. Sample the new adjacency matrix $A' \sim A^\dagger$.

We apply this framework to the adjacency matrix A (no transformation), the random walk transition matrix $P = D^{-1}A$, the symmetric normalized Laplacian $L^{\text{sym}} = I - D^{-\frac{1}{2}}A^{-\frac{1}{2}}$, and the modularity matrix $B = A - dd^\top/(2e(G))$. The rank of the approximation is chosen so that the desired edge overlap with the input graph is reached: on CORA-ML, we use rank 1600 for $M \in \{A, P, B\}$ and rank 2520 for $M = L^{\text{sym}}$. For the transition matrix, we choose the same back-transformation $\tilde{A} = \text{diag}(\tilde{\pi})\tilde{P}$ as in our method instead of $\tilde{A} = D\tilde{P}$, which uses the degree matrix of the input graph. Given $\tilde{A}$, we proceed like NetGAN and our method with $S = \tilde{A}$. For link prediction, we also use the score matrix.

- **Low-rank approximation with respect to cross-entropy loss.** This baseline is a version of our method CELL, but without the logit space. That is, we solve the optimization problem

$$\begin{aligned} \min_{\tilde{P} \in \mathbb{R}^{N\times N}} & -\sum_{k,l=1}^{N} A_{k,l} \log \tilde{P}_{k,l}\,, \\ \text{s.t.} \quad & \text{rank}(\tilde{P}) \leqslant H \quad \text{and} \quad \tilde{P} \in \mathcal{P}\,, \end{aligned} \tag{17}$$

where $\mathcal{P}$ is the set of stochastic matrices on $\mathbb{R}^{N\times N}$. Similar to how we proceed in CELL, we enforce this constraint with the parametrization

$$\tilde{P} = D(e^C e^D)^{-1} e^C e^D\,, \tag{18}$$

where $C \in \mathbb{R}^{N\times H}$, $D \in \mathbb{R}^{H\times N}$, the exponential function is taken element-wise, and $D(e^C e^D)$ denotes the diagonal matrix of row sums for $e^C e^D$. We then optimize the objective in Eq. (17) over C and D with Adam.

C.3. Stopping criteria

In addition to the rank constraint, CELL and NetGAN both use an early stopping criterion for learning the random walk distribution.

EO-criterion. The EO-criterion is the stopping criterion used in this paper and by NetGAN, and generates graphs with a predefined edge overlap with the input graph (e. g. 50%). To employ it, training is stopped during regular intervals, the edge-independent model is constructed, and a single graph G' is generated. If the fraction of edges $e(G \cap G')/e(G)$ is smaller than the predefined threshold, training is continued, otherwise it is stopped.

VAL-criterion. The VAL-criterion is a stopping criterion proposed by NetGAN and represents an alternative to the edge overlap (EO) criterion used in this paper. It is employed by evaluating the link prediction performance on the validation set during training of the optimization problem, and then stopping the training as soon as the link prediction performance does not improve for a predefined amount of training iterations. For evaluation after training, the test set is used instead of the validation set.

C.4. Example of a bias of NetGAN and CELL: ε-neighborhood graphs

This section demonstrates the concepts discussed in Section 5.2 on ε-neighborhood graphs. These graphs arise by choosing the nodes as points in a metric space, and connecting those pairs of points by an unweighted, undirected edge whose distance is smaller than a constant $\varepsilon > 0$ (see Figure 3 for an illustration). Given an ε-graph as input, there is one major property that a graph generative model should keep intact: edges should occur between points with a small distance in the underlying space, but not for points with a large distance. Figure 3 shows that NetGAN and CELL do not comply with this desired tendency: both algorithms generate long edges. We can easily counteract this mismatch in inductive bias for CELL by extending our loss function with an additional term that penalizes long edges. Because the underlying distances of the metric space are not directly represented in the ε-neighborhood graph anymore, we use its shortest path distances $\mathcal{D} \in \mathbb{R}^{N\times N}$ as a proxy (which is sensible as one can prove that the shortest path distances in ε-graphs converge to the underlying metric

distances (Tenenbaum et al., 2000; Orlitsky, 2005)). We refer to the resulting method as "Local CELL", whose loss function is given by

$$F(W) = -\sum_{k,l=1}^{N} A_{k,l} \log \sigma_{\text{rows}}(W)_{k,l} - \sum_{k,l=1}^{N} A_{k,l} \left[\mathcal{D}_{k,l} \leqslant k\right] \log \sigma_{\text{rows}}(W)_{k,l} \,, \tag{19}$$

where $[\mathcal{A}] = 1$, if statement $\mathcal{A}$ is true, and 0 otherwise. A comparison of NetGAN, CELL, and Local CELL is given in Table 5, see also Figure 3 for an illustration. As expected, NetGAN and CELL generate graphs with long edges, resulting in a large average edge length and small characteristic path length. Local CELL on the other hand is significantly closer to the input graph in this regard without a loss in performance for other statistics. It even improves on some other statistics, because the objective function is more appropriate for this type of graph.

Table 5. Statistics of ε-neighborhood graph and generated graphs from three generative models, averaged over five trials. For the generated graphs, "avg. edge len." is computed only from generated edges that are not present in the input graph.

GRAPH		CHARAC. PATH LEN.	AVG. EDGE LEN.	SPECTRAL GAP	ASSORT-ATIVITY	POWER LAW EXP.	TRIANGLE COUNT	WEDGE COUNT
ε-NEIGHBORHOOD GRAPH		8.97	0.10	2.04e-3	0.75	1.51	2,319	11,557
NETGAN	(51% EO)	3.54	0.41	4.49e-2	0.09	1.50	895	10,638
CELL	(52% EO)	4.01	0.50	3.57e-2	0.38	1.50	1,144	11,030
LOCAL CELL	(52% EO)	5.92	0.21	5.13e-3	0.47	1.51	1,088	11,377

C.5. Additional baseline experiments

Graph statistics and link prediction performance on all data sets described in Section 6 for generated graphs from NetGAN, our method CELL, and baselines, averaged over five trials. Statistics that are matched by model design for the configuration model are indicated as $*$, and cases that are not applicable as $-$. For a visualization and interpretation of the results, see Section C.7.

Table 6. CORA-ML (2,810 nodes, 7,981 edges).

GRAPH		MAX. DEGREE	ASSORT-ATIVITY	TRIANGLE COUNT	SQUARE COUNT	POWER LAW EXP.	CLUSTER-ING COEFF.	CHARAC. PATH LEN.	ROC-AUC SCORE	TIME (IN S)
CORA-ML		238	-0.076	2,802	14,268	1.86	8.26e-2	5.63	1	–
CONF. MODEL	(52% EO)	$*$	-0.053	623	3111	$*$	1.96e-2	4.43	–	1
LR-ADJ	(53% EO)	121	-0.042	444	1,128	1.72	2.78e-2	5.17	0.561	32
LR-TRANS	(57% EO)	139	-0.058	558	1,617	1.77	2.94e-2	5.07	0.709	33
LR-LAP	(52% EO)	167	-0.084	691	1942	1.79	2.79e-2	4.76	0.800	38
LR-MOD	(53% EO)	122	-0.043	437	1,135	1.72	2.75e-2	5.17	0.557	48
LR-CE	(52% EO)	193	-0.068	1,388	6,284	1.79	5.68e-2	5.37	0.950	73
NETGAN	(54% EO)	219	-0.071	1,461	5,555	1.80	5.23e-2	5.13	0.950	7,478
CELL	(53% EO)	204	-0.070	1,396	6,880	1.82	5.07e-2	5.26	0.938	21

Table 7. CITESEER (2,110 nodes, 3,668 edges).

GRAPH	MAX. DEGREE	ASSORT-ATIVITY	TRIANGLE COUNT	SQUARE COUNT	POWER LAW EXP.	CLUSTER-ING COEFF.	CHARAC. PATH LEN.	ROC-AUC SCORE	TIME (IN S)
CITESEER	72	-0.015	483	1,866	2.24	8.70e-2	10.68	1	–
CONF. MODEL (56% EO)	∗	-0.014	108	282	∗	1.95e-2	6.33	–	1
LR-ADJ (57% EO)	34	4.75e-2	89	188	2.09	2.62e-2	8.17	0.608	12
LR-TRANS (57% EO)	36	-0.022	119	364	2.15	3.20e-2	8.58	0.825	8
LR-LAP (57% EO)	48	0.019	108	161	2.18	2.45e-2	7.82	0.362	12
LR-MOD (56% EO)	32	5.33e-4	87	162	2.09	2.67e-2	8.17	0.603	108
LR-CE (56% EO)	47	-0.076	138	549	2.13	3.50e-2	8.57	0.903	19
NETGAN (57% EO)	52	-0.074	361	478	2.15	8.50e-2	9.03	0.951	4,654
CELL (56% EO)	44	-0.093	106	318	2.17	2.54e-2	7.36	0.858	10

Table 8. POLBLOGS (1,222 nodes, 16,779 edges).

GRAPH	MAX. DEGREE	ASSORT-ATIVITY	TRIANGLE COUNT	SQUARE COUNT	POWER LAW EXP.	CLUSTER-ING COEFF.	CHARAC. PATH LEN.	ROC-AUC SCORE	TIME (IN S)
POLBLOGS	298	-0.222	60,873	2,631,731	1.44	0.189	2.82	1	–
CONF. MODEL (52% EO)	∗	-0.140	31,364	1,263,826	∗	0.118	2.72	–	1
LR-ADJ (52% EO)	171	-0.022	15,497	430,846	1.36	0.082	2.66	0.63	1
LR-TRANS (52% EO)	200	-0.114	27,428	918,543	1.40	0.114	2.73	0.861	1
LR-LAP (51% EO)	234	-0.214	19,593	511,781	1.36	0.086	2.55	0.745	2
LR-MOD (51% EO)	170	-0.028	15,528	433,669	1.36	0.082	2.66	0.624	16
LR-CE (54% EO)	248	-0.226	34,942	1,303,305	1.40	0.126	2.66	0.943	17
NETGAN (52% EO)	261	-0.244	37,849	1,438,174	1.41	0.132	2.70	0.950	55,276
CELL (51% EO)	268	-0.243	49,366	2,043,407	1.43	0.160	2.78	0.949	15

Table 9. RT-GOP (4,687 nodes, 5,529 edges).

GRAPH	MAX. DEGREE	ASSORT-ATIVITY	TRIANGLE COUNT	SQUARE COUNT	POWER LAW EXP.	CLUSTER-ING COEFF.	CHARAC. PATH LEN.	ROC-AUC SCORE	TIME (IN S)
RT-GOP	270	-0.135	0	2	4.29	0	14.01	1	–
CONF. MODEL (51% EO)	∗	-0.092	56	241	∗	1.89e-3	5.68	–	1
LR-ADJ (52% EO)	239	-0.117	0	87	3.74	0	12.05	0.559	7
LR-TRANS (55% EO)	328	-0.111	0	139	4.53	0	6.18	0.676	19
LR-LAP (52% EO)	162	-0.070	5	1	3.09	5.15e-4	14.61	0.466	164
LR-MOD (51% EO)	192	-0.101	0	34	3.41	0	21.10	0.550	84
LR-CE (51% EO)	233	-0.122	0	29	3.74	0	20.08	0.874	129
NETGAN (52% EO)	221	-0.112	14	15	3.64	6.74e-4	16.33	0.738	14,800
CELL (51% EO)	253	-0.142	0	6	4.11	0	16.90	0.704	23

Table 10. WEB-EDU (3,031 nodes, 6,547 edges).

GRAPH		MAX. DEGREE	ASSORT-ATIVITY	TRIANGLE COUNT	SQUARE COUNT	POWER LAW EXP.	CLUSTER-ING COEFF.	CHARAC. PATH LEN.	ROC-AUC SCORE	TIME (IN S)
WEB-EDU		99	-0.183	4491	35,423	2.11	0.167	4.56	1	–
CONF. MODEL	(52% EO)	∗	-0.109	873	4,913	∗	0.032	4.59	–	1
LR-ADJ	(53% EO)	58	-0.114	1,096	3,932	1.97	0.081	6.66	0.579	18
LR-TRANS	(53% EO)	166	-0.034	2,692	20,424	2.13	0.120	5.13	0.862	12
LR-LAP	(53% EO)	103	-0.098	514	1,637	2.01	0.028	5.25	0.360	30
LR-MOD	(53% EO)	58	-0.121	989	3,292	1.97	0.075	6.44	0.595	97
LR-CE	(52% EO)	74	-0.136	1,194	4,330	1.99	0.080	6.41	0.994	72
NETGAN	(53% EO)	92	-0.174	1,244	3,022	2.02	0.064	5.51	0.992	11,000
CELL	(54% EO)	63	-0.234	1,176	4,710	2.03	0.069	6.67	0.977	16

C.6. Evolution of graph statistics during training

To compare the generated graphs of CELL and NetGAN for different edge overlaps, we fix all hyperparameters and stop training at regular intervals to compute the graph statistics of the generated graphs. Since we fix the ranks for the low-rank constraint, the generated graphs will not converge to 100% edge overlap. But this area of high edge overlap is of little interest anyway, because the shared edges alone force the generated graphs to reproduce many graph statistics. Note that in order to reduce computational complexity, NetGAN uses less random walks to compute statistics during training (for example to evaluate the stopping criteria): instead of sampling many random walks from the generator, it keeps track of the random walks generated in the last 1,000 iterations during training to build the score matrix. For our method CELL there is no such distinction, we complete our pipeline as described in the main paper.

Aside from few exceptions, for example the triangle count and the related clustering coefficient on CITESEER, we observe the same behavior as described in Section 6.2 of the main paper: after a short initialization phase, CELL and NetGAN display comparable behavior.

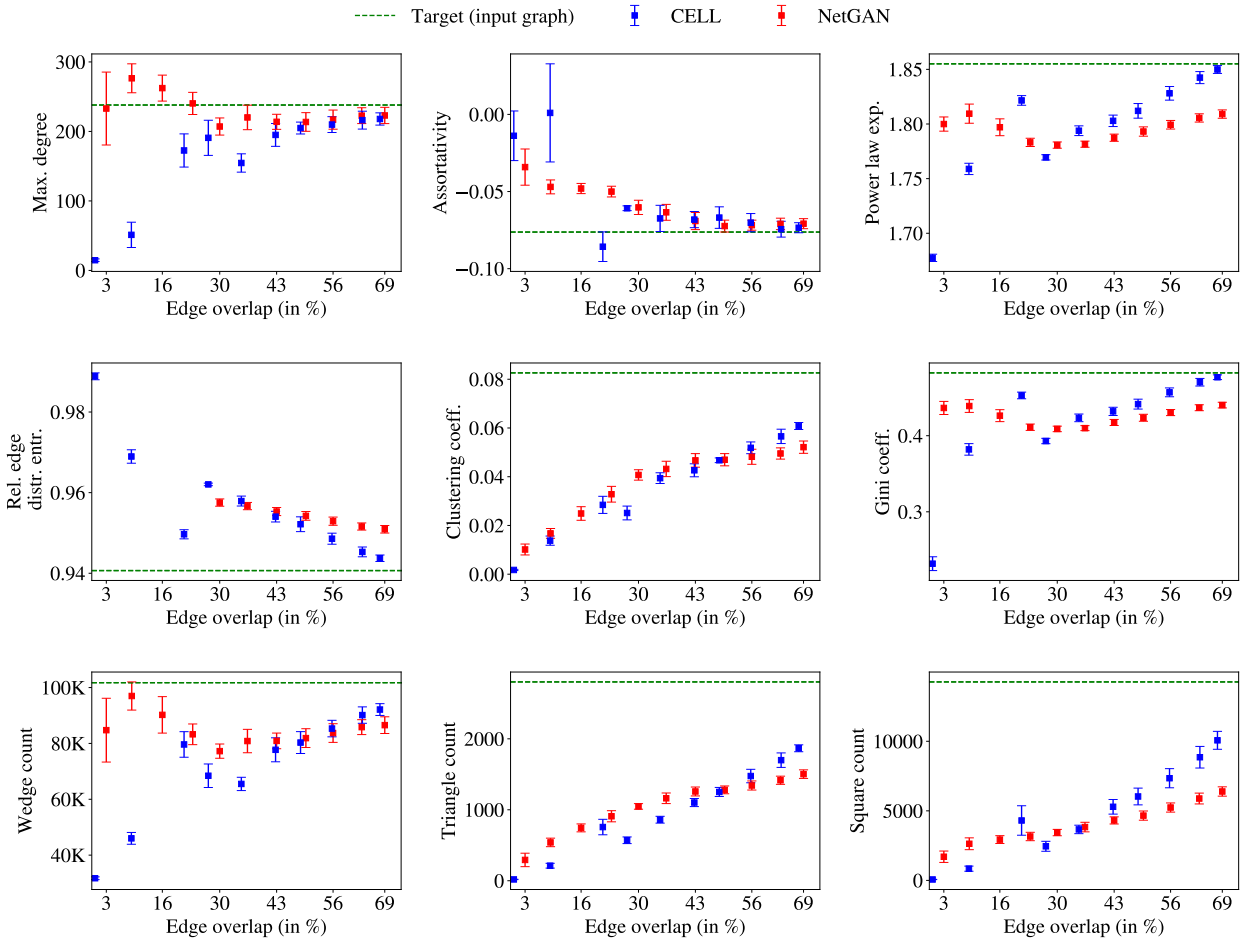

Figure 5. Graph statistics during training for NetGAN and CELL on CORA-ML, plotted against edge overlap.

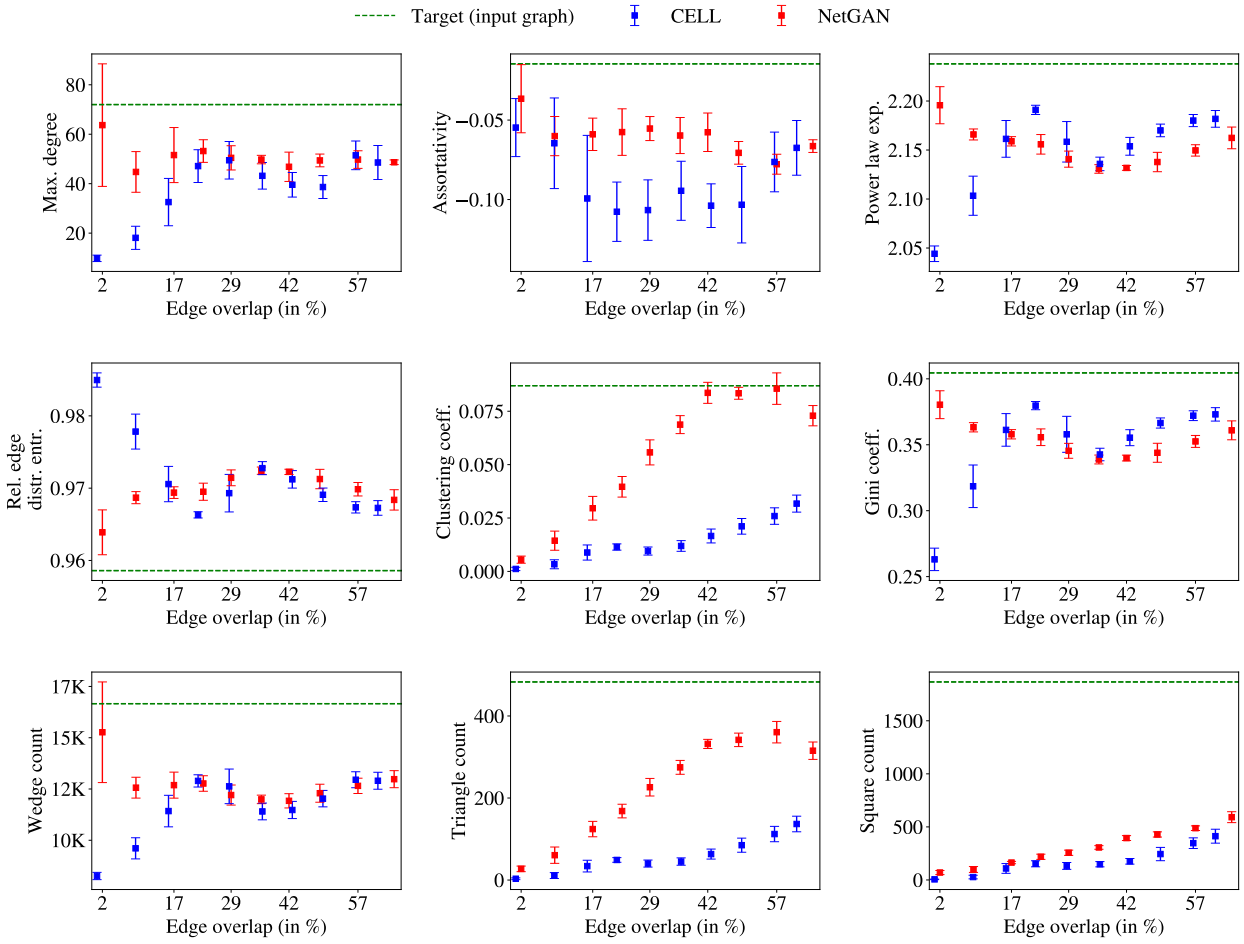

Figure 6. Graph statistics during training for NetGAN and CELL on CITESEER, plotted against edge overlap.

C.7. Comparison of relative errors

For a graph statistic s on the input graph, let $s_{\mathrm{M}} = (s_{\mathrm{M}}^{(1)}, \ldots, s_{\mathrm{M}}^{(K)})$ denote the estimates of model M for s in K trials. The average relative error is then defined by $s_{\mathrm{rel}}(\mathrm{M}) = 1/K \sum_{k=1}^{K} \left| s - s_{\mathrm{M}}^{(k)} \right| / |s|$. In Figure 7, we depict the relative errors for NetGAN, CELL, and baselines on a variety of data sets and graph statistics. Small relative errors indicate good performance in the sense that the generated graphs are close to the input graph. Over all data sets, a general trend can be observed: on most instances, the three models NetGAN, CELL, and LR-CE behave similarly and better as compared to the other baselines. Occasional deviations of this behavior might be attributed to the different optimization procedures and the early stopping. For some networks, for example CITESEER, NetGAN seems to outperform CELL, but for others like POLBLOGS, CELL performs better; this reflects that their different optimization procedures might or might not contribute to the goal of learning the network at hand.

C.8. Hyperparameters

For all our considered models except the configuration model and NetGAN, we choose the rank parameter H such that the generated graphs achieve the predefined edge overlap with the input graph. For example on CORA-ML with 2,810 nodes, we choose $H = 1600$ for LR-Adj, LR-Trans, and LR-Mod, $H = 2520$ for LR-Lap, $H = 950$ for LR-CE, and only $H = 9$ for our method CELL. In general, a higher rank increases the ability of the model to generate graphs with a high edge overlap. For NetGAN, we only consider unbiased random walks ($p = q = 1$) with batch size 128 and length 16. The dimensions H_g and H_d for the low-rank projection for generator and discriminator are both 128. Both generator and discriminator have a single hidden layer with 40 hidden units for the generator and 30 hidden units for the discriminator. The temperature τ is annealed from $\tau = 5$ to $\tau = 0.5$ with a multiplicative decay of $1 - 10^{-5}$ every step.

We optimize the methods LR-CE, NetGAN and CELL using Adam. For LR-CE and CELL, we use a learning rate of 0.1 and weight decay of 10^{-7}, and for NetGAN the learning rate is 0.0003 with L_2-regularization of 10^{-7} for the generator and $5 \cdot 10^{-5}$ for the discriminator. The Wasserstein gradient penalty is set to 10.

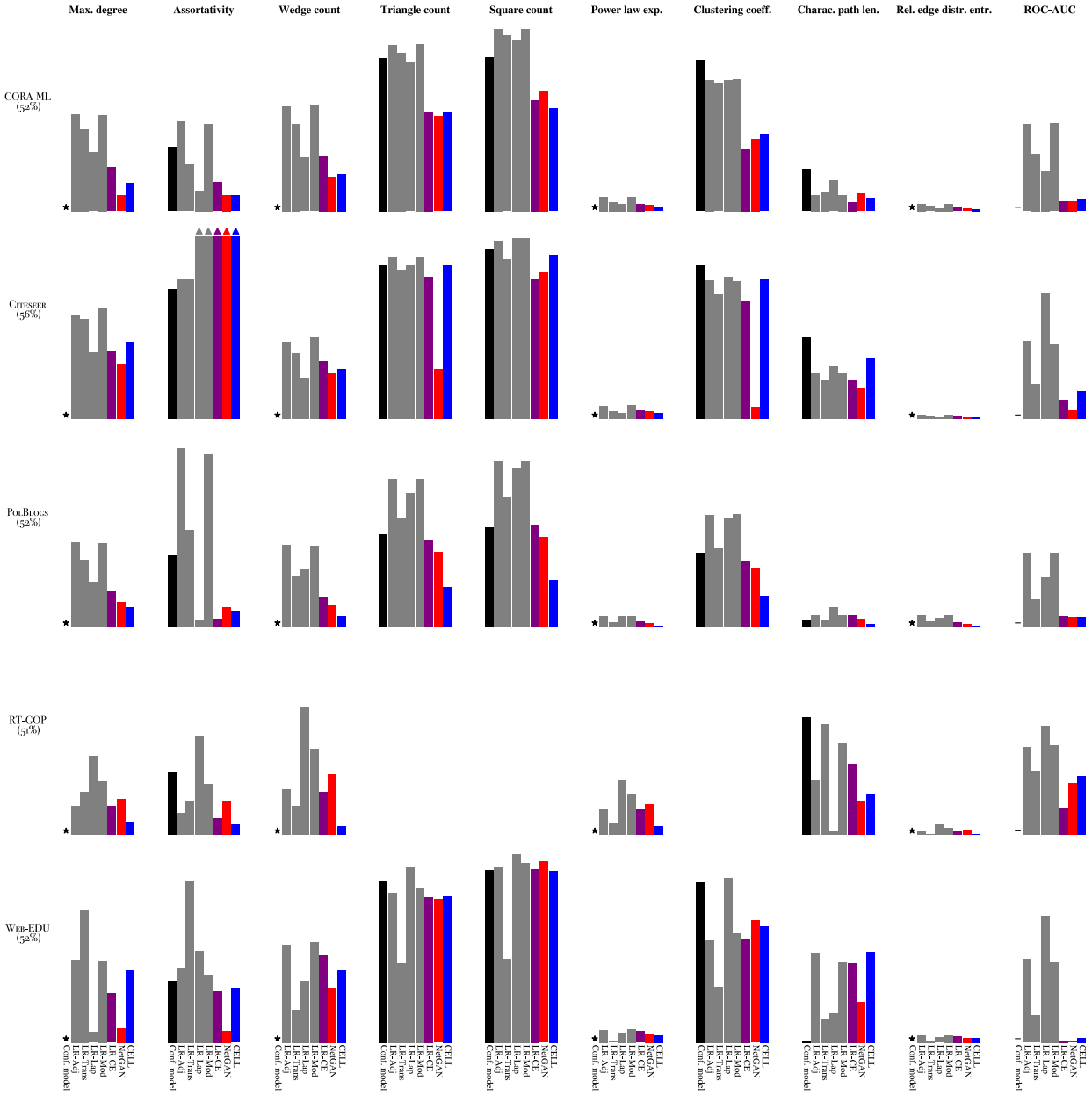

Figure 7. Relative errors of NetGAN, CELL and baselines, trained until the EO-stopping criterion given in brackets, and averaged over five trials. Rows represent the input graphs, columns represent the graph statistics. The y-axis ranges from 0 to 1 in every cell; values that exceed 1 are capped and indicated with an arrow. For the Conf. model, statistics that are matched exactly (0 relative error) are indicated by $\star$, and the non-existent ROC-AUC score is indicated by a $-$. The three statistics triangle count, square count and clustering coefficient for the extremely sparse network RT-GOP are omitted, because their value is zero and the relative errors are not defined (triangle count, clustering coefficient), or it is too small to be produce a meaningful relative error (square count is 2). For the actual graph statistics of generated and input graphs, see Section C.5.