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

Numerical Linear Algebra course final project

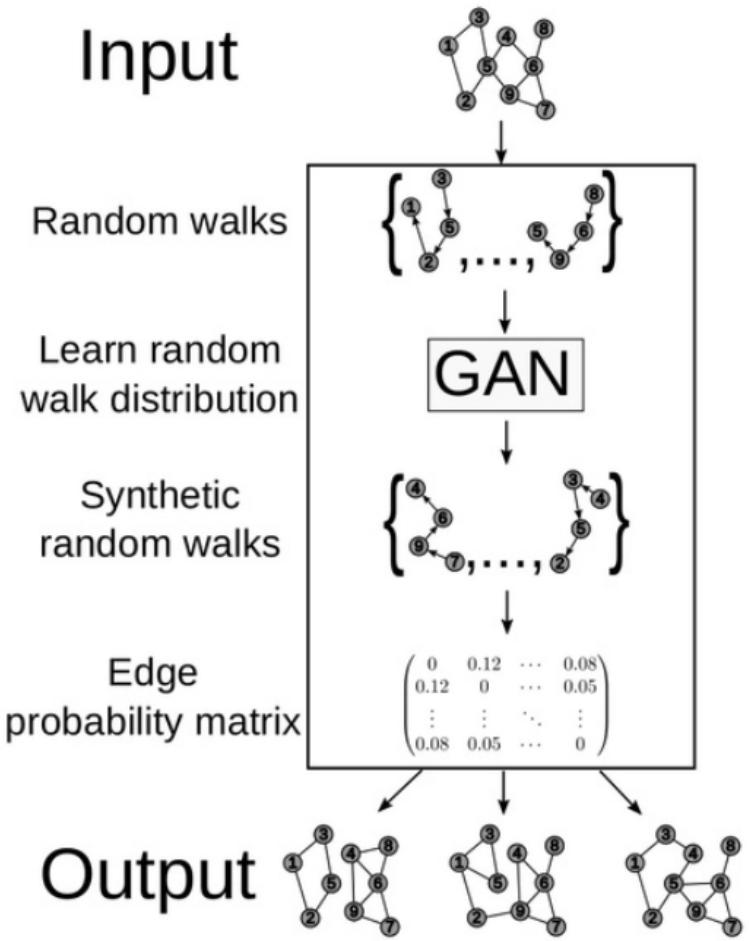
Evgeny Lagutin, Pavel Burnyshev, Valentin Samokhin, Daniil Moskovskiy

December 19, 2020

Introduction

NetGAN aims to reconstruct graphs similar to the input one. NetGAN algorithm in short:

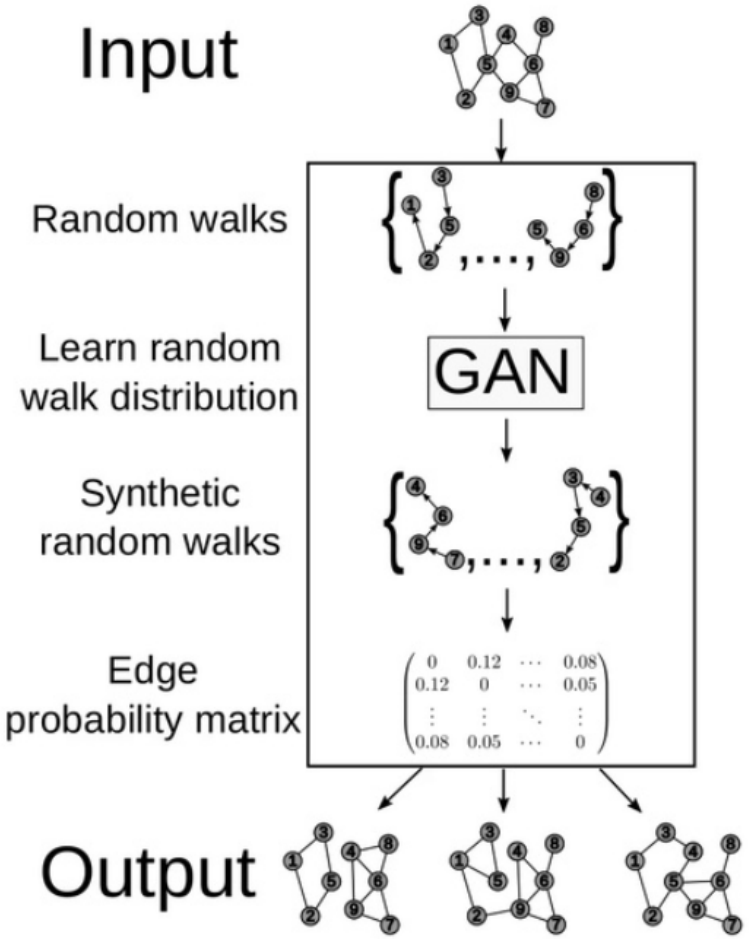
1. Sample a set of random walks from the input graph to train the GAN
2. Learn random walk distribution
3. Sample a set of random walks from the GAN
4. Construct edge probability matrix from the produced set



Introduction

NetGAN aims to reconstruct graphs similar to the input one. NetGAN algorithm in short:

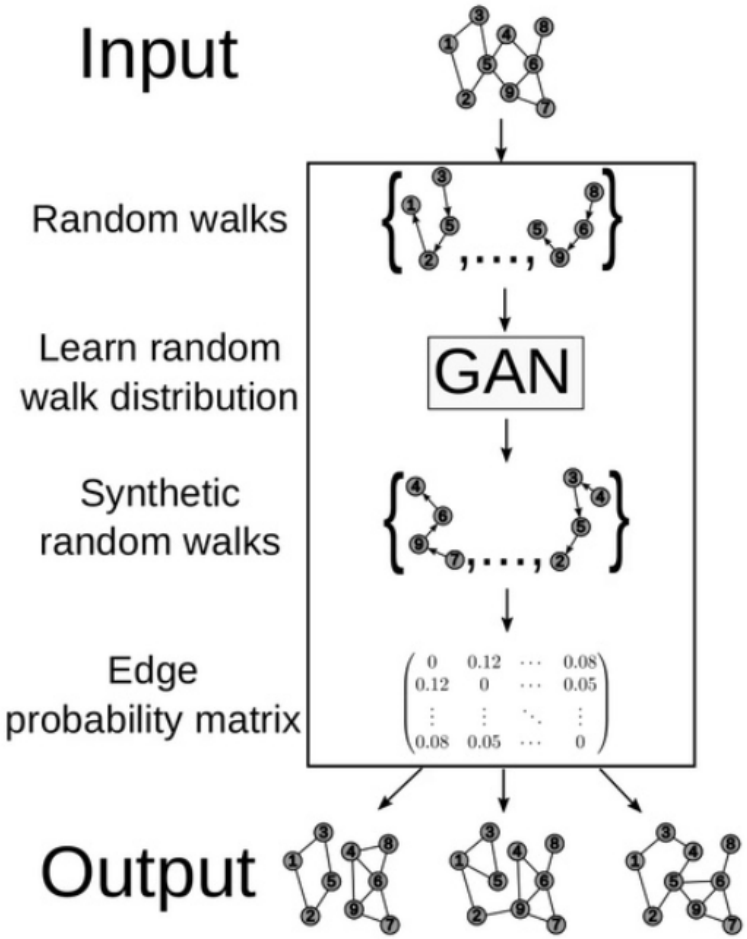
1. Sample a set of random walks from the input graph to train the GAN
2. Learn random walk distribution
3. Sample a set of random walks from the GAN
4. Construct edge probability matrix from the produced set



Introduction

NetGAN aims to reconstruct graphs similar to the input one. NetGAN algorithm in short:

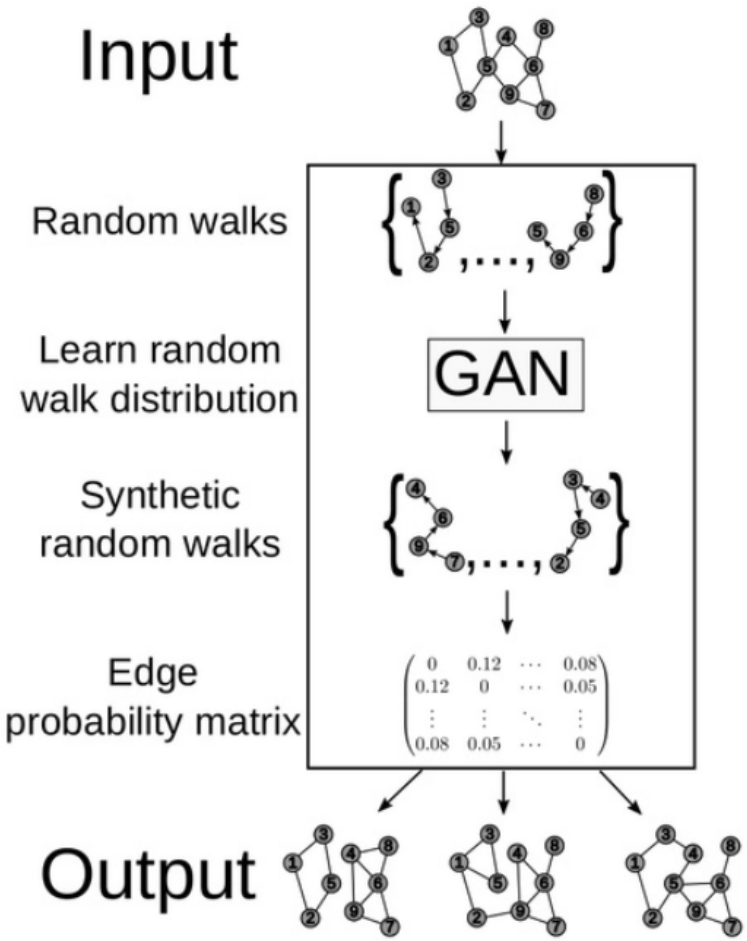
1. Sample a set of random walks from the input graph to train the GAN
2. Learn random walk distribution
3. Sample a set of random walks from the GAN
4. Construct edge probability matrix from the produced set



Introduction

NetGAN aims to reconstruct graphs similar to the input one. NetGAN algorithm in short:

1. Sample a set of random walks from the input graph to train the GAN
2. Learn random walk distribution
3. Sample a set of random walks from the GAN
4. Construct edge probability matrix from the produced set

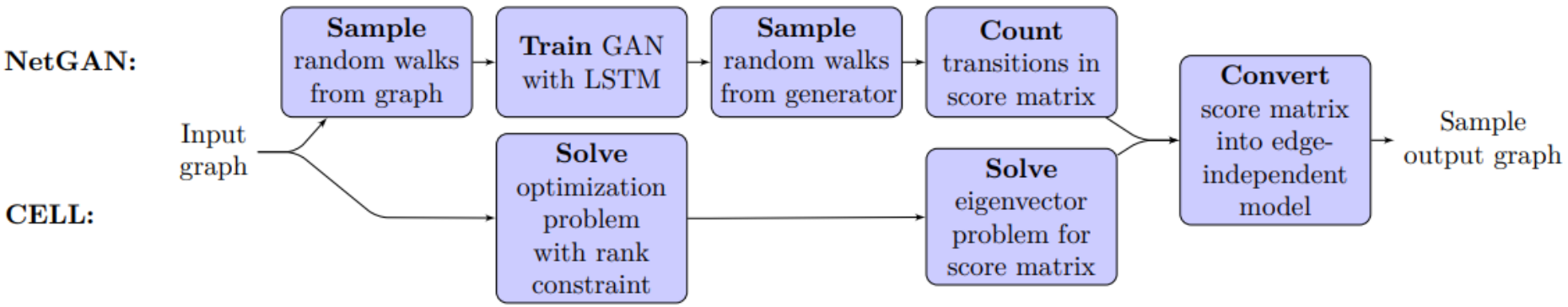


NetGAN

Main goal: Learn random walk distribution from a training set based on unbiased random walks $\{v_0^i, \ldots, v_T^i\}_i$ over input graph. It's supposed to be connected and non-bipartite.
Generator generates sequences $\{\omega_0^i, \ldots, \omega_T^i\}_i$ of "synthetic" random walks while **discriminator** tries to distinguish "synthetic" random walks from the real ones.
Core model is LSTM with Wasserstein loss (therefore, no long-term dependencies due to LSTM usage, and comparable slow training). Update equations for **synthetic** random walk are given by:

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

NetGAN without GAN approach



- Replaces the original problem to a low-rank approximation with respect to the Kullback-Leibler divergence between transition matrices (requires neither a GAN nor any sampling).
- Increases the speed of training and is more transparent due to simpler structure
- Evades random walk sampling part which was (will be shown further) very computationally expensive and inefficient.

Replacing the GAN

Skolkovo Institute of Science and Technology

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, the generator f_θ can be written as

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

where $g_\theta : \mathbb{R}^H \to \mathbb{R}^H$ 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 $W(m_t) := \widetilde{W}_{\text{down}}(m_t) W_{\text{up}} \in \mathbb{R}^{N \times N}$ to obtain

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

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

Replacing the GAN

The authors propose dropping the LSTM with it the dependency on the memory state m_t; this is justified by the Markov property of unbiased r.w.:

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

Authors suggest to drop the structural restriction imposed by g_θ, setting $W = W_{\text{down}}\ W_{\text{up}}$ and update rule

$$p_{t+1} = v_t^\top W_{down} W_{up}$$

But we may leave the g_θ part by rewriting as (v_θ is one-hot vector):

$$p_{t+1} = v_t^\top g_\theta(W_{down}) W_{up}$$

Therefore $v_{t+1} \sim \text{Cat}\left(\sigma\left(W_{v_t}\right)\right)$.
Now training the GAN is equivalent to learning the r.w. transition matrix from the family $\mathcal{P} = \left\{\sigma_{\text{rows}}\left(W\right) \in \mathbb{R}^{N\times N} : W \in \mathbb{R}^{N\times N}, \text{rank}(W) \leqslant H\right\}$

No GAN no problem

Consider a graph with number of nodes N, set of random walks $\mathcal{R}$, low rank H row-wise softmax σ_{rows}
Learn random walk distribution by **learning random walk transition matrix** from parametric family:

$$\mathcal{P} = \{\sigma_{\text{rows}}(W) \in \mathbb{R}^{N \times N} : W \in \mathbb{R}^{N \times N},\ \text{rank}(W) \leq H\}$$

Solve an optimization problem: $\begin{cases} -\sum\limits_{(i,j) \in \mathcal{R}} \log \sigma_{\text{rows}}(W)_{i,j} \to \min\limits_{W \in \mathbb{R}^{N \times N}} \\ \text{s.t.: } \text{rank}(W) \leq H \end{cases}$

Latent variable z plays the subordinate role of choosing the first node.

Evade random walk sampling

NetGAN samples a huge amount of random walks from generator that learned the distribution from the input graph
E.g., for CORA-ML graph (2810 nodes, 7891 edges) NetGAN samples that much random walks, that every edge of the graph is processed for 14000 times (!).
Given length of random walks T, number of random walks n, score matrix $S \in \mathbb{R}^{N \times N}$, the normalized score matrix converges:

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

where P is corresponding random walk transition matrix and π is stationary distribution.

NetGAN without GAN

Learning step with adjacency matrix A

$$\begin{cases} -\sum\limits_{(i,j)\in\mathcal{R}} \log \sigma_{\text{rows}}(W)_{i,j} \to \min\limits_{W\in\mathbb{R}^{N\times N}} \\ \text{s.t.: } \text{rank}(W) \leq H \end{cases} \quad \begin{cases} -\sum\limits_{k,l} A_{k,l} \log \sigma_{\text{rows}}(W)_{k,l} \to \min\limits_{W\in\mathbb{R}^{N\times N}} \\ \text{s.t.: } \text{rank}(W) \leq H \end{cases}$$

Reconstruction step (with synthetic transition matrix P^*):

1. Find π^* s.t. $\pi^{*\top} = \pi^{*\top} P^*$
2. Take $S := \text{diag}(\pi^*) P^*$

No sampling during both steps!!!

Cross-Entropy Low-rank Logits (CELL)

Algorithm 1: CELL algorithm

Input: adjacency matrix $A \in \{0,1\}^{N \times N}$, rank $H << N$

Output: matrix of edge probabilities $A^\dagger \in [0,1]^{N \times N}$

1. Solve optimization problem for W^* (shown on the previous slide)
2. Compute transition matrix: $P^* \longleftarrow \sigma_{\text{rows}}(W^*)$
3. Solve eigenvalue problem: $\pi^{*T} P^* = \pi^{*T}$ for π^*
4. Compute score matrix: $S \longleftarrow \text{diag}(\pi^*) P^*$
5. Convert score matrix S to edge-independent model $A^\dagger$:
   $S^\dagger \leftarrow \max\{S, S^T\}$; $A^\dagger \leftarrow S^\dagger / \sum_{i,j} S^\dagger_{i,j}$, so
   $A^\dagger_{i,j} = \max\{S_{i,j}\} / \sum_{k',l'=1}^{N} \max\{S_{k,l}, S_{k',l'}\}$
6. Return $A^\dagger$

Graph Statistics Used

Assortativity	$\frac{\text{cov}X,Y}{\sigma_X \sigma_Y}$
Power Law Exponent	$1 + n\left(\sum_{v \in V} \log \frac{d(v)}{d_{\min}}\right)^{-1}$
Relative edge distr. entropy	$-\frac{1}{\log N} \sum_{v \in V} \frac{d(v)}{2m} \log \frac{d(v)}{2m}$
Gini coeff.	$\frac{2\sum_{i=1}^{N} i\hat{d}_i}{\sum_{i=1}^{N} \hat{d}_i} - \frac{N+1}{N}$
Character. path length	$\frac{1}{N(N-1)} \sum_{uv} d(u,v)$
Spectral gap	$\lambda_1(L)$
Motif count	$--$

Table: Graph statistics used

The performance is measured by the ROC-AUC score, applied to the score matrix evaluated at the edges in question (test edges)

Our contribution

- replication of the proposed approach and results
- extending parametric family of low rank matrices W with decomposition on non-linear transformations (implementing W as output of fully-connected auto-encoder with I_N as input)
- experimenting with low-rank approximations of W with SVD by explicitly constrating U, V to be close to orthogonal during gradient descent

Experiments and results

Method	d_{max}	Assort.	Triangle count	Square count	Power Law Exp	Clustering Coeff.	ROC-AUC
Original Graph	246	-0.077	5247	34507	1.77	0.114	1
CELL	213	-0.066	1235	5936	**1.81**	0.045	0.94
SVD	196	**-0.079**	1264	6813	1.83	0.046	0.95
nonlinear-CELL	**245**	-0.074	**1791**	**10122**	1.86	**0.054**	**0.96**

Figure: Results on CORA-ML graph (2810 nodes, 7891 edges) using edge overlap criterion.

Method	d_{max}	Assort.	Triangle count	Square count	Power Law Exp	Clustering Coeff.	ROC-AUC
Original Graph	99	0.060	1083	5977	2.068	0.125	1
CELL	55	-0.060	138	413	**2.183**	0.031	0.88
SVD	60	-0.084	165	501	2.228	0.033	**0.92**
nonlinear-CELL	**96**	**0.017**	**441**	**3041**	2.330	**0.062**	0.88

Figure: Results on Citeseer graph (2110 nodes, 3668 edges) using edge overlap criterion.

Experiments and results

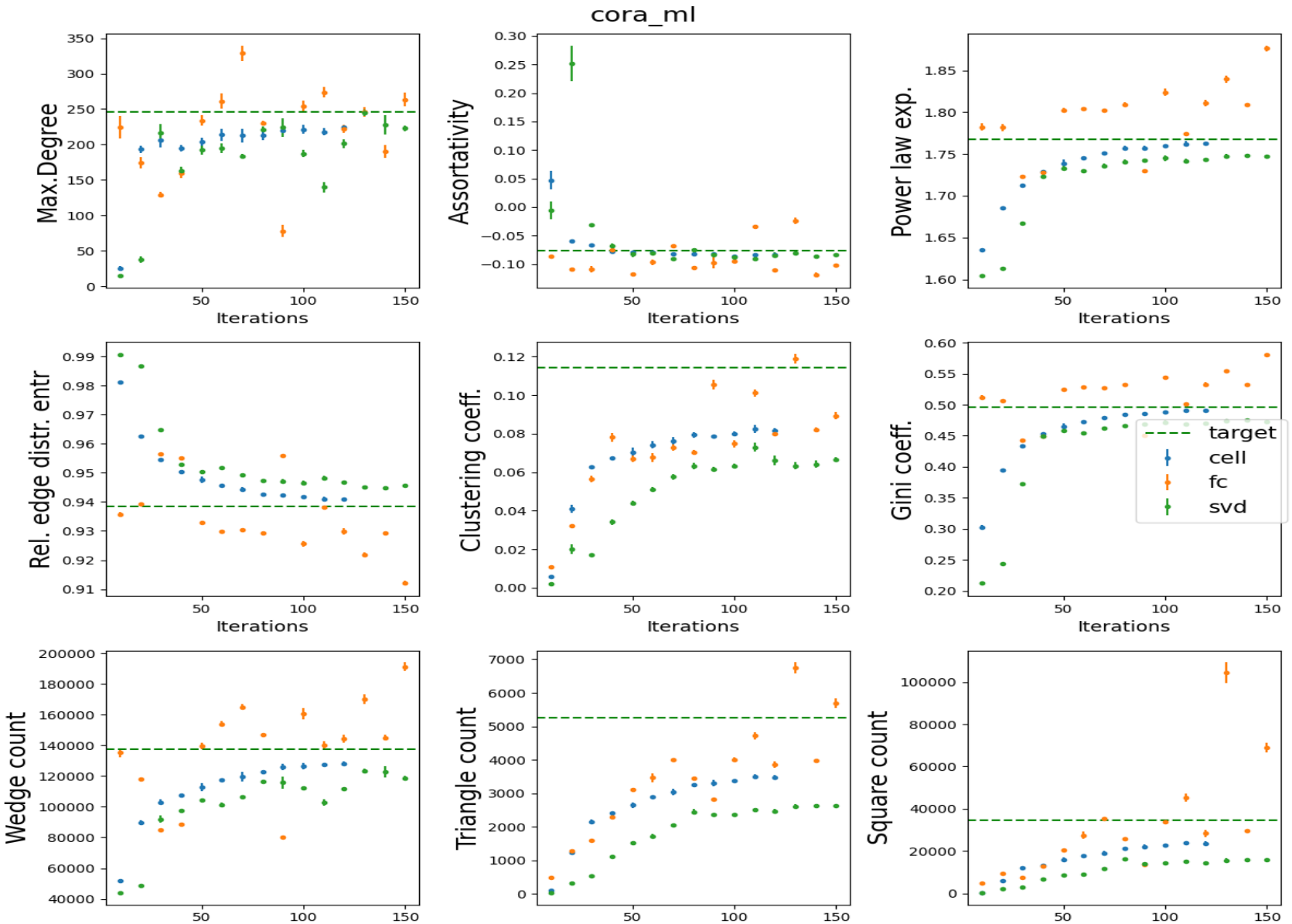
Method	d_{max}	Assort.	Triangle count	Square count	Power Law Exp	Clustering Coeff.	ROC-AUC
Original Graph	351	-0.184	101043	5171257	1.414	0.226	1
CELL	260	**-0.225**	40880	1576069	1.407	0.144	**0.96**
SVD	234	-0.231	**43161**	**1725619**	**1.412**	**0.149**	**0.96**
nonlinear-CELL	**276**	-0.229	42507	1696934	1.419	0.148	0.95

Figure: Results on PolBlogs graph (1222 nodes, 16779 edges) using edge overlap criterion.

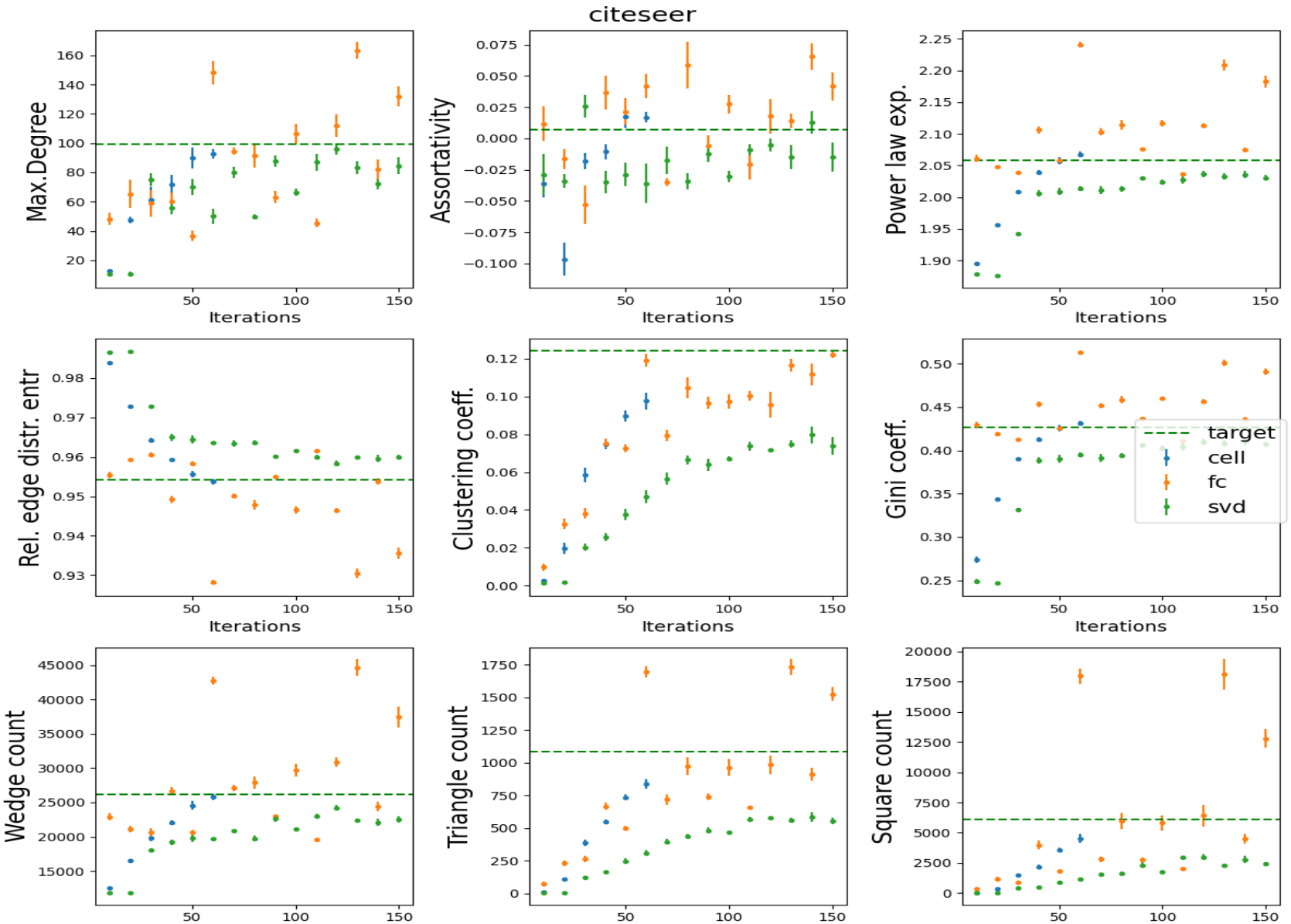
Method	d_{max}	Assort.	Triangle count	Square count	Power Law Exp	Clustering Coeff.	ROC-AUC
Original Graph	308	-0.090	33	369	3.493	$0.0008 \approx 0$	1
CELL	**248**	-0.142	0	11	**3.942**	0	0.69
SVD	234	**-0.137**	0	10	4.149	0	0.77
nonlinear-CELL	242	-0.142	0	**13**	4.269	0	**0.84**

Figure: Results on RT-GOP graph (4687 nodes, 5529 edges) using edge overlap criterion.

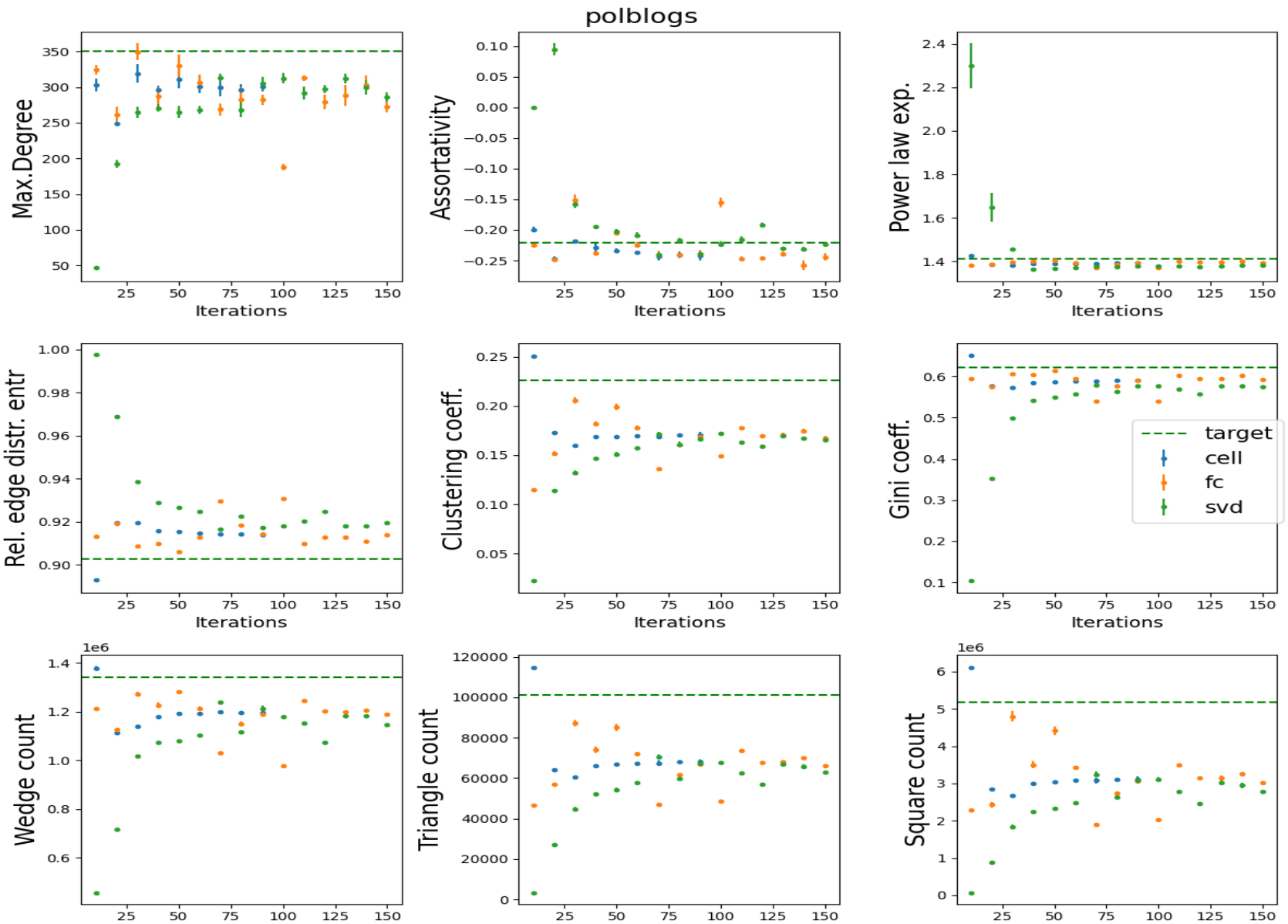
Experiments: plots - CORAML



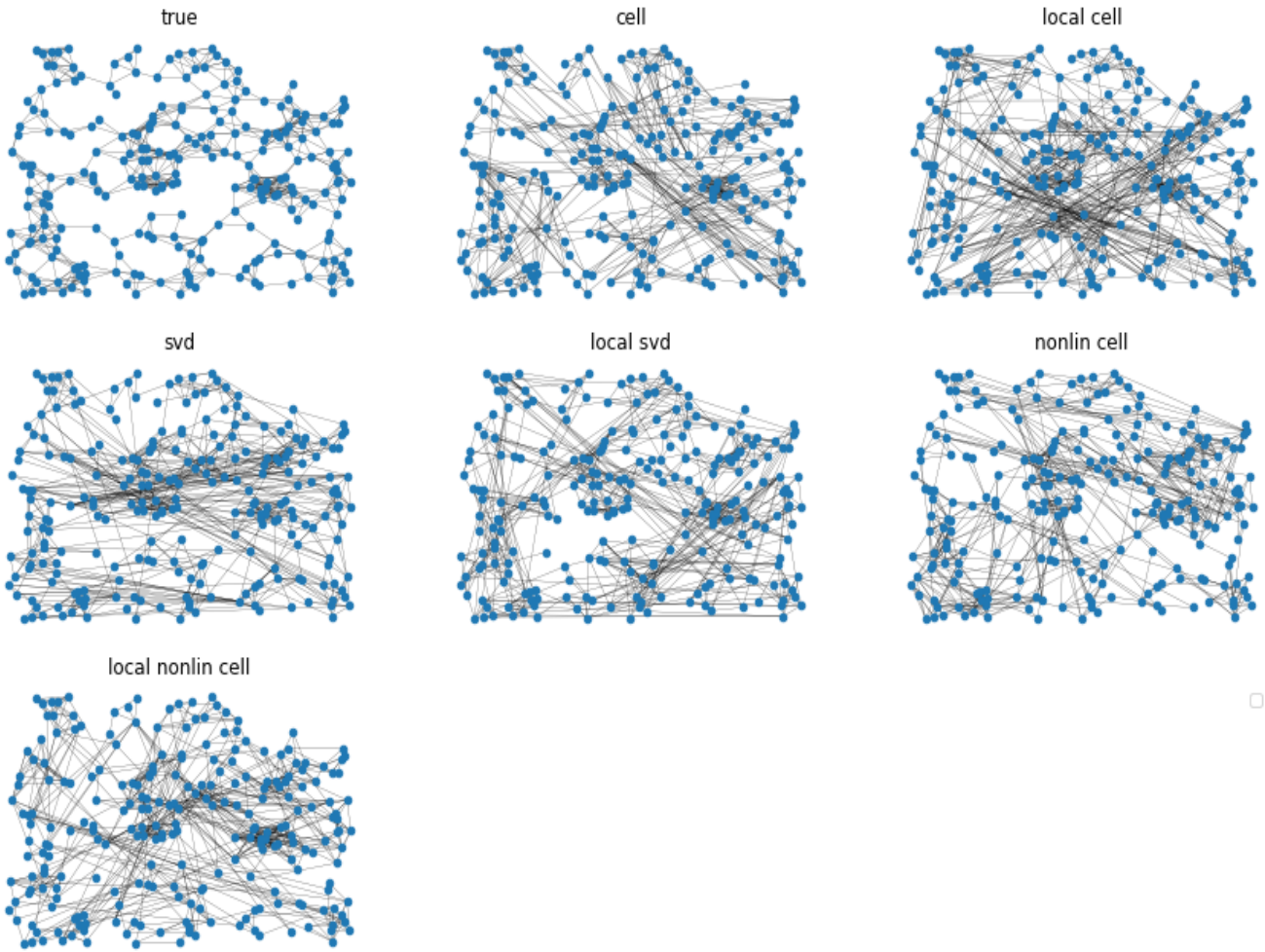
Experiments: plots - Citeseer



Experiments: plots - PolBlogs



Experiments: ε-neighbourhood graph



- Thank you for attention!