GraphRNN: A Deep Generative Model for Graphs (24 Feb 2018)

Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, Jure Leskovec
Presented by: Jesse Bettencourt and Harris Chan

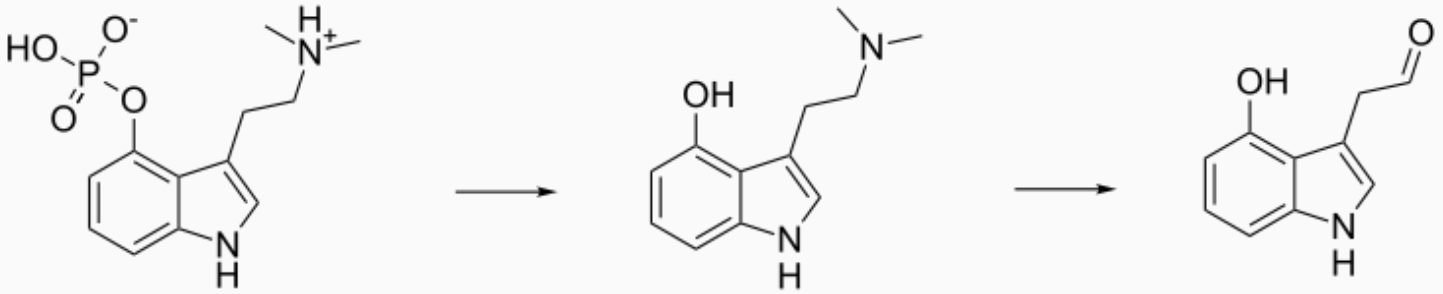
March 9, 2018

University of Toronto, Vector Institute

Introduction: Generative Model for Graphs

Modeling graphs is fundamental for studying networks

e.g. medical, chemical, social



Goal:

Model and efficiently sample complex distributions over graphs

Learn generative model from observed set of graphs

Challenges in Graph Generation

Large and variable output spaces

Graph with n nodes requires n^2 to fully specify structure

Number of nodes and edges varies between different graphs

Non-unique representations

Distributions over graphs without assuming fixed set of nodes

n node graph represented by up to $n!$ equivalent adjacency matrices

$\pi \in \Pi$ is arbitrary node ordering

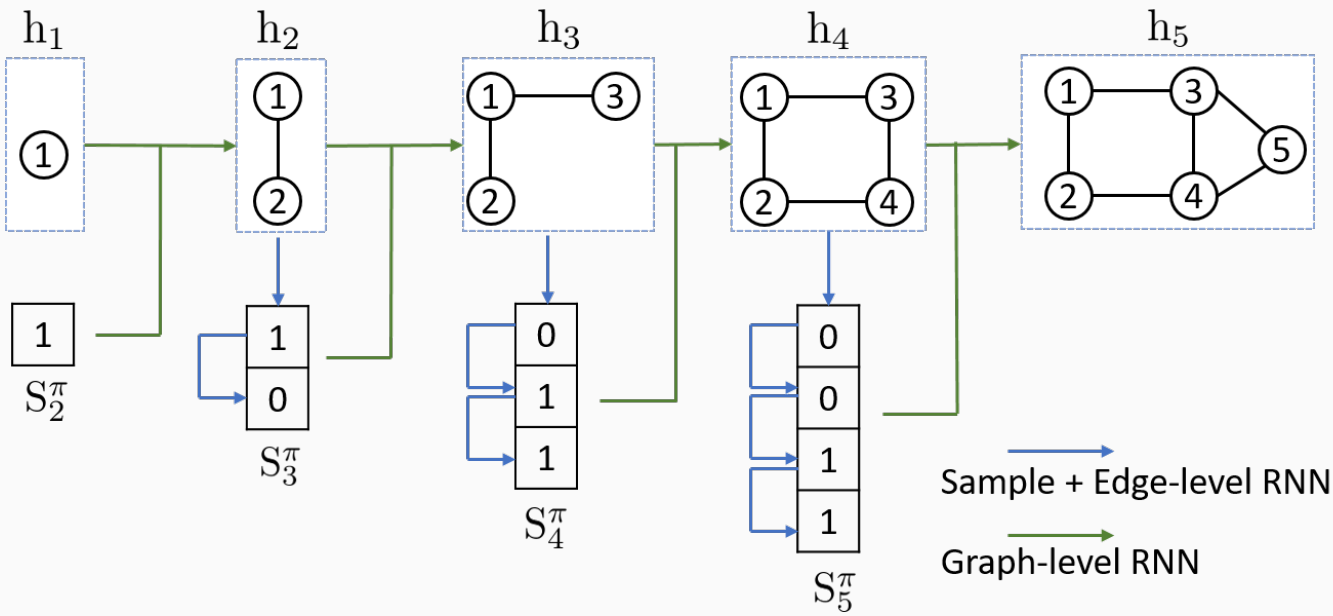
Complex, non-local dependencies

New edges depend on previously generated edges

Overview to GraphRNN

Decompose graph generation into two RNNs:

- Graph-level: generates sequence of nodes
- Edge-level: generates sequence of edges for each new node



Modeling Graphs as Sequences

Graph $G \sim p(G)$ with n nodes under node ordering π

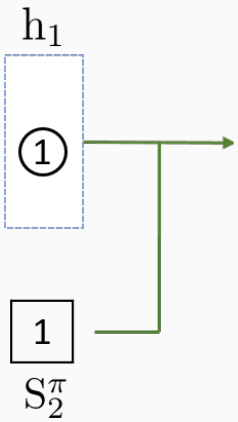
Define mapping f_s from G to sequence

$$S^\pi = f_S(G, \pi) = (S_1^\pi, \ldots, S_n^\pi) \tag{1}$$

Each sequence element is adjacency vector

$$S_i^\pi \in \{0,1\}^{i-1} \quad i \in \{1, \ldots, n\}$$

for edges between node $\pi(v_i)$ and $\pi(v_j)$, $j \in \{1, \ldots, i-1\}$



Sample + Edge-level RNN

Graph-level RNN

Modeling Graphs as Sequences

Graph $G \sim p(G)$ with n nodes under node ordering π

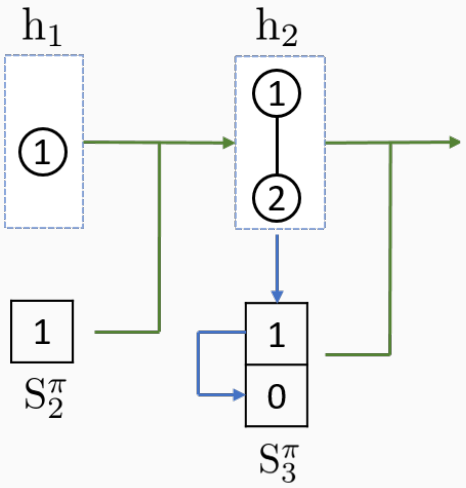
Define mapping f_s from G to sequence

$$S^\pi = f_S(G, \pi) = (S_1^\pi, \ldots, S_n^\pi) \tag{1}$$

Each sequence element is adjacency vector

$$S_i^\pi \in \{0,1\}^{i-1} \quad i \in \{1, \ldots, n\}$$

for edges between node $\pi(v_i)$ and $\pi(v_j)$, $j \in \{1, \ldots, i-1\}$



Sample + Edge-level RNN

Graph-level RNN

Modeling Graphs as Sequences

Graph $G \sim p(G)$ with n nodes under node ordering π

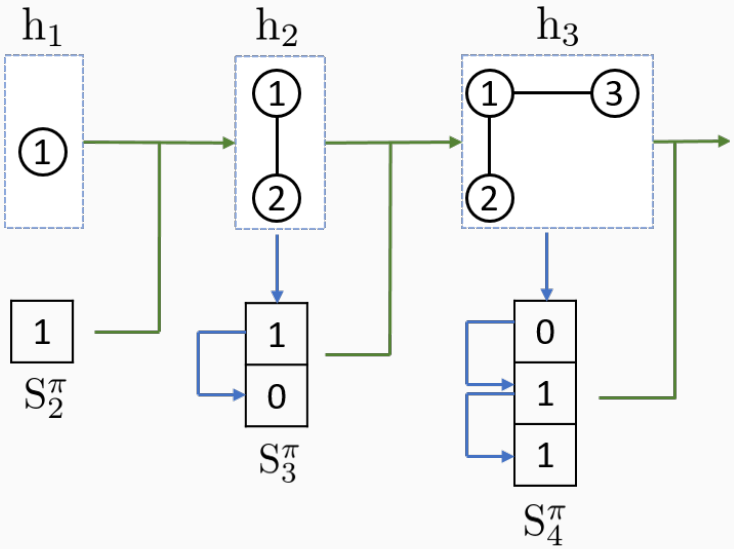
Define mapping f_s from G to sequence

$$S^\pi = f_S(G, \pi) = (S_1^\pi, \ldots, S_n^\pi) \tag{1}$$

Each sequence element is adjacency vector

$$S_i^\pi \in \{0,1\}^{i-1} \quad i \in \{1, \ldots, n\}$$

for edges between node $\pi(v_i)$ and $\pi(v_j)$, $j \in \{1, \ldots, i-1\}$



Sample + Edge-level RNN

Graph-level RNN

Modeling Graphs as Sequences

Graph $G \sim p(G)$ with n nodes under node ordering π

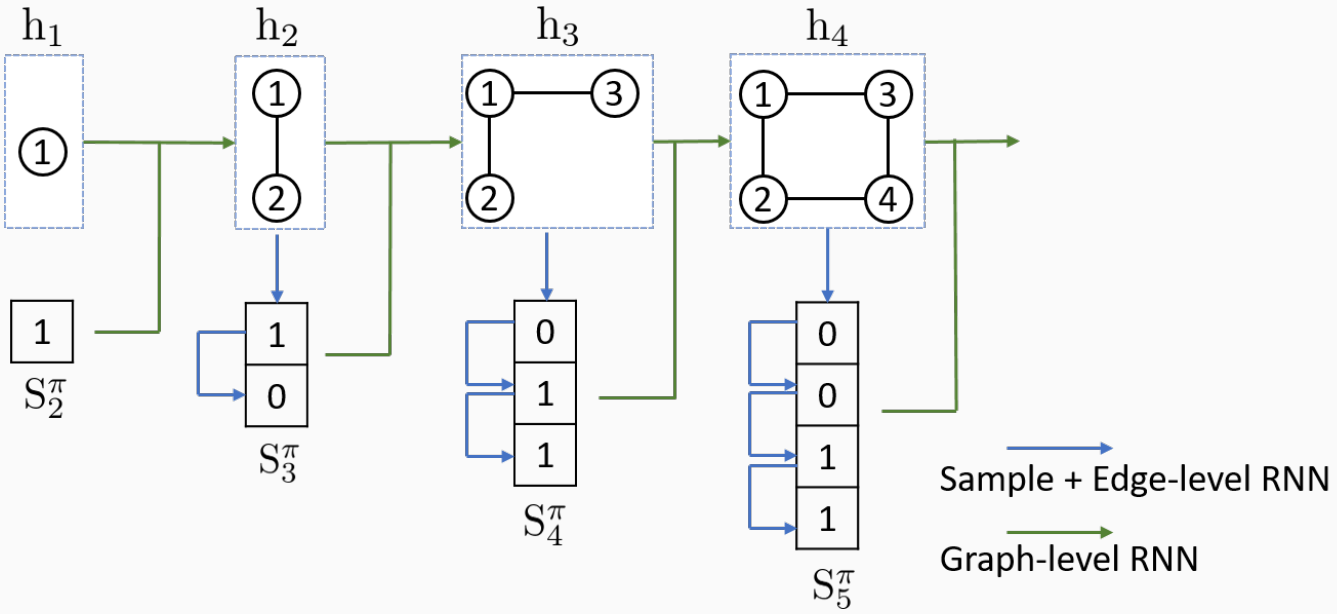
Define mapping f_s from G to sequence

$$S^\pi = f_S(G, \pi) = (S_1^\pi, \ldots, S_n^\pi) \tag{1}$$

Each sequence element is adjacency vector

$$S_i^\pi \in \{0,1\}^{i-1} \quad i \in \{1, \ldots, n\}$$

for edges between node $\pi(v_i)$ and $\pi(v_j)$, $j \in \{1, \ldots, i-1\}$



Modeling Graphs as Sequences

Graph $G \sim p(G)$ with n nodes under node ordering π

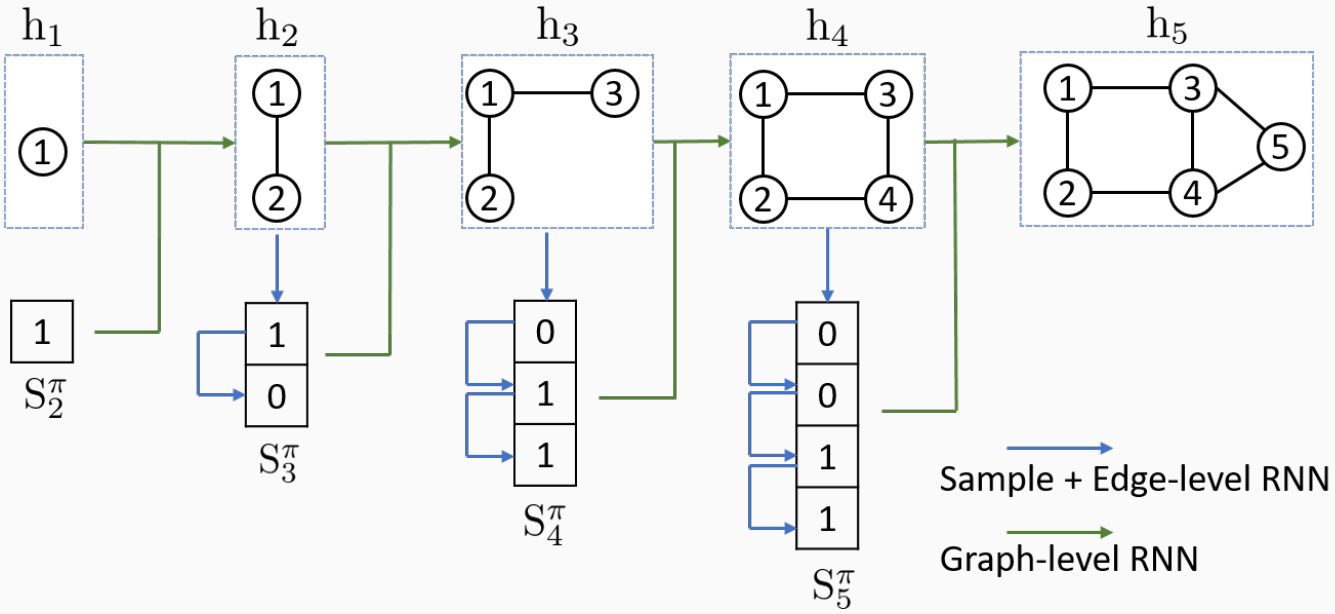
Define mapping f_s from G to sequence

$$S^\pi = f_S(G, \pi) = (S_1^\pi, \ldots, S_n^\pi) \tag{1}$$

Each sequence element is adjacency vector

$$S_i^\pi \in \{0,1\}^{i-1} \quad i \in \{1, \ldots, n\}$$

for edges between node $\pi(v_i)$ and $\pi(v_j)$, $j \in \{1, \ldots, i-1\}$



Distribution on Graphs $\rightarrow$ Distribution on Sequences

Instead of learning $p(G)$ sample, $\pi \sim \Pi$ to get observations of S^π
Then learn $p(S^\pi)$ modeled autoregressively:

$$p(G) = \sum_{S^\pi} p(S^\pi) \mathbb{1}[f_G(S^\pi) = G] \tag{3}$$

Exploiting sequential structure of S^π, decompose $p(S^\pi)$

$$\begin{aligned} P(S^\pi) &= \prod_{i=1}^{n+1} p(S_i^\pi | S_1^\pi, \ldots, S_{i-1}^\pi) \\ &= \prod_{i=1}^{n+1} p(S_i^\pi | S_{<i}^\pi) \end{aligned} \tag{4}$$

Motivating GraphRNN

Model $p(G)$

Distribution over graphs

$\downarrow$

Model $p(S^{\pi})$

Distribution over sequence of edge connections

$\downarrow$

Model $p(S_i^{\pi}|S_{<i}^{\pi})$

Distribution over edge connections for i-th node conditioned on previous nodes' edge connections

parameterize with an expressive neural network

GraphRNN Framework

Idea: Use an RNN that consists of a *state-transition* function and an *output* function:

$$h_i = f_{\text{trans}}(h_{i-1}, S_{i-1}^{\pi}) \tag{5}$$

$$\theta_i = f_{\text{out}}(h_i) \tag{6}$$

- $h_i \in \mathbb{R}^d$ encodes the state of the graph generated so far
- S_{i-1}^{π} encodes adjacency for most recently generated node $i-1$
- θ_i specifies the distribution of next node's adjacency vector

$$S_i^{\pi} \sim \mathcal{P}_{\theta_i}$$

- f_{trans} and f_{out} can be arbitrary neural networks
- $\mathcal{P}_{\theta_i}$ can be an arbitrary distribution over binary vectors

GraphRNN Framework Corrected

Idea: Use an RNN that consists of a *state-transition* function and an *output* function:

$$h_i = f_{\text{trans}}(h_{i-1}, S_i^\pi) \tag{5}$$

$$\theta_{i+1} = f_{\text{out}}(h_i) \tag{6}$$

- $h_i \in \mathbb{R}^d$ encodes the state of the graph generated so far
- S_i^π encodes adjacency for most recently generated node i
- θ_{i+1} specifies the distribution of next node's adjacency vector

$$S_{i+1}^\pi \sim \mathcal{P}_{\theta_{i+1}}$$

- f_{trans} and f_{out} can be arbitrary neural networks
- $\mathcal{P}_{\theta_i}$ can be an arbitrary distribution over binary vectors.

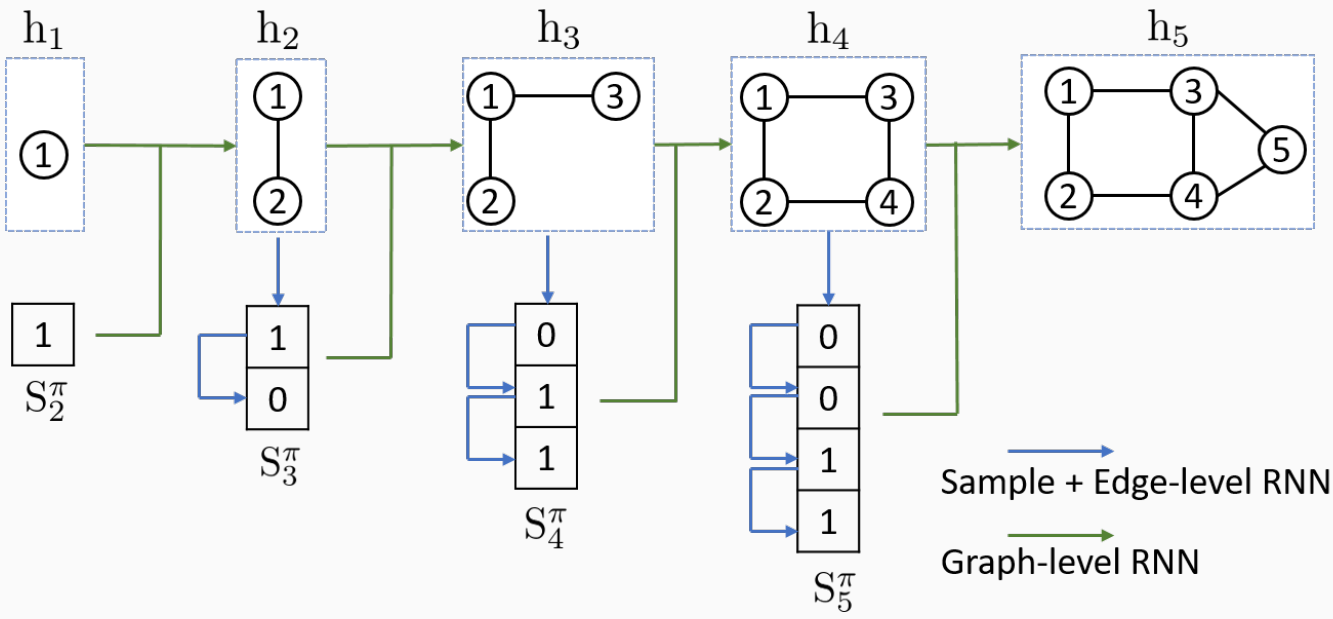
GraphRNN Framework Corrected

Idea: Use an RNN that consists of a *state-transition* function and an *output* function:

$$h_i = f_{\text{trans}}(h_{i-1}, S_i^\pi) \tag{5}$$

$$\theta_{i+1} = f_{\text{out}}(h_i) \tag{6}$$

$$S_{i+1}^\pi \sim \mathcal{P}_{\theta_{i+1}}$$



GraphRNN Inference Algorithm

Algorithm 1 GraphRNN inference algorithm

Input: RNN-based transition module f_{trans}, output module f_{out}, probability distribution $\mathcal{P}_{\theta_i}$ parameterized by θ_i, start token SOS, end token EOS, empty graph state h'

Output: Graph sequence S^{π}

$S_0^{\pi} = \text{SOS}$, $\boldsymbol{h}_0 = h'$, $i = 0$

repeat

$i = i + 1$

$h_i = f_{\text{trans}}(h_{i-1}, S_{i-1}^{\pi})$ {update graph state}

$\theta_i = f_{\text{out}}(h_i)$

$S_i^{\pi} \sim \mathcal{P}_{\theta_i}$ {sample node i's edge connections}

until S_i^{π} is EOS

Return $S^{\pi} = (S_1^{\pi}, ..., S_i^{\pi})$

GraphRNN Inference Algorithm Corrected

Algorithm 1 GraphRNN inference algorithm

Input: RNN-based transition module f_{trans}, output module f_{out}, probability distribution $\mathcal{P}_{\theta_i}$ parameterized by θ_i, start token SOS, end token EOS, empty graph state h'

Output: Graph sequence S^{π}

$S^{\pi}_{0 \to 1} = \text{SOS}$, $\boldsymbol{h}_0 = h'$, $i = 0$

repeat

$i = i + 1$

$h_i = f_{\text{trans}}(h_{i-1}, S^{\pi}_{i-1 \to i})$ {update graph state}

$\theta_{i \to i+1} = f_{\text{out}}(h_i)$

$S^{\pi}_{i \to i+1} \sim \mathcal{P}_{\theta_{i \to i+1}}$ {sample node $i \to i+1$'s edge connections}

until $S^{\pi}_{i \to i+1}$ is EOS

Return $S^{\pi} = (S^{\pi}_1, ..., S^{\pi}_i)$

GraphRNN Variants

Objective: $\prod p_{model}(S^\pi)$ over all observed graph sequences
Implement f_{trans} as **Gated Recurrent Unit (GRU)**
But different assumptions about $p(S_i^\pi | S_{<i}^\pi)$ for each variant:

1. **Multivariate Bernoulli** (GraphRNN-S):
   f_{out} is a MLP with sigmoid activation that outputs $\theta_{i+1} \in \mathbb{R}^i$
   θ_{i+1} parameterizes the multivariate Bernoulli
   $S_{i+1}^\pi \sim \mathcal{P}_{\theta_{i+1}}$ *independently*

GraphRNN Variants

Objective: $\prod p_{model}(S^{\pi})$ over all observed graph sequences
Implement f_{trans} as **Gated Recurrent Unit (GRU)**
But different assumptions about $p(S_i^{\pi}|S_{<i}^{\pi})$ for each variant:

2. **Dependent Bernoulli sequence** (GraphRNN):

$$p(S_i^{\pi}|S_{<i}^{\pi}) = \prod_{j=1}^{i-1} p(S_{i,j}^{\pi}|S_{i,<j}^{\pi}, S_{<i}^{\pi}) \tag{7}$$

- $S_{i,j}^{\pi} \in \{0, 1\}$ indicating if node $\pi(v_i)$ is connected to node $\pi(v_j)$
- f_{out} is a *edge-level* RNN generates the edges of a given node

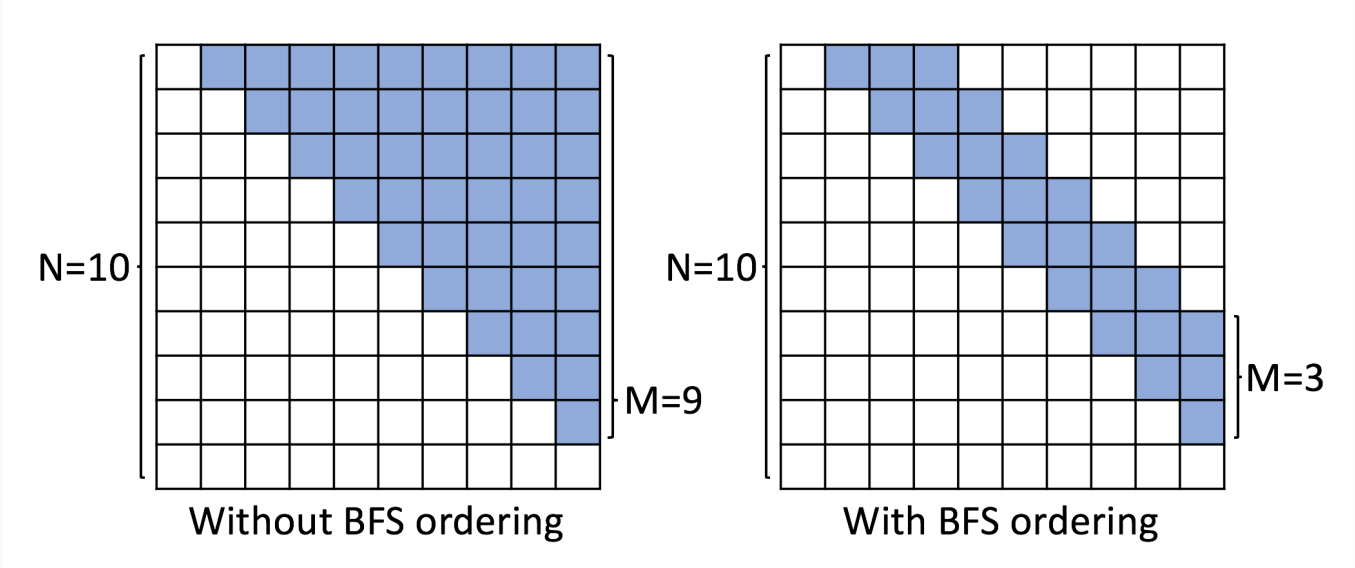
Tractability via Breadth First Search (BFS)

Idea: Apply BFS ordering to the graph G with node permutation π before generating the sequence S^{π}

Benefits:

- **Reduce overall # of seq to consider**
  Only need to train on all possible BFS orderings, rather than all possible node permutations
- **Reduce the number of edge predictions**
  Edge-level RNN only predicts M edges, the maximum size of the BFS queue

BFS Order Leads To Fixed Size S_i^π



$S_i^\pi \in \mathbb{R}^M$ represents "sliding window" over nodes in the BFS queue
Zero-pad all S_i^π to be a length M vector:

$$S_i^\pi = (A^\pi_{\max(1,i-M),i}, ..., A^\pi_{i-1,i})^T, i \in \{2, ..., n\} \quad (9)$$

Experiments

Datasets

3 Synthetic and 2 real graph datasets:

Dataset	Type	# Graphs	Graph Size	Description
Community	Synthetic	500	$60 \leq \|\|V\|\| \leq 160$	2-community, Erdős-Rényimodel (E-R)
Grid	Synthetic	100	$100 \leq \|V\| \leq 400$	Standard 2D grid
B-A	Synthetic	500	$100 \leq \|V\| \leq 200$	Barabási-Albert model, 4 existing nodes connected
Protein	Real	918	$100 \leq \|V\| \leq 500$	Amino acids nodes, edge if $\leq$ 6 Angstroms apart
Ego	Real	757	$50 \leq \|V\| \leq 399$	Document nodes, edges citation relationships, from Citeseer

Baseline Methods & Settings

- Compared GraphRNN to traditional models and deep learning baselines:

Method Type	Algorithm
Traditional	Erdős-RényiModel (E-R) (Erdös & Rényi, 1959)
	Barabási-Albert Model (B-A) (Albert & Barabási, 2002)
	Kronecker graph models (Leskovec et al., 2010)
	Mixed-membership stochastic block models (MMSB) (Airoldi et al., 2008)
Deep learning	GraphVAE (Simonovsky & Komodakis, 2018)
	DeepGMG (Li et al., 2018)

- 80%-20% train-test split
- All models trained with early stopping
- Traditional methods: learn from a single graph, so train a separate model for each training graph in order to compare with these methods
- Deep learning baselines: use smaller dataset:
  Community-small: $12 \leq |V| \leq 20$
  Ego-small: $4 \leq \|V\| \leq 18$

Evaluating Generated Graph Via MMD Metric

Existing:

- Visual Inspection
- Simple comparisons of average statistics between the two sets

Proposed:

A metric based on **Maximum Mean Discrepancy (MMD)**, to compare all moments of their empirical distributions using an exponential kernel with Wasserstein distance.

Graph Visualization

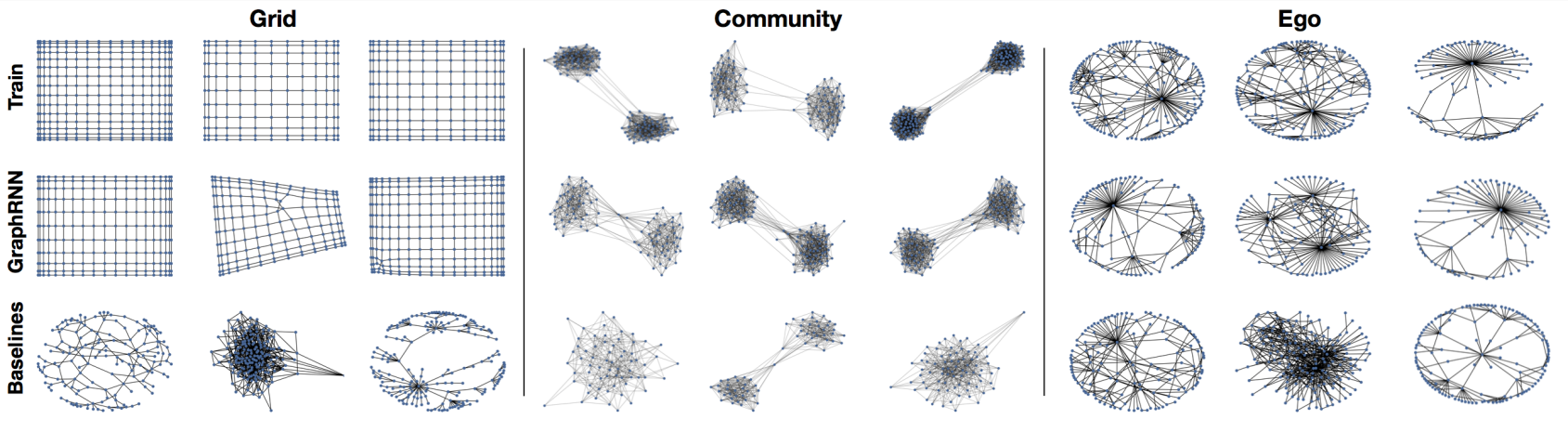


Figure 2: Visualization of graphs from grid dataset (Left group), community dataset (Middle group) and Ego dataset (Right group). Within each group, graphs from training set (First row), graphs generated by GraphRNN(Second row) and graphs generated by Kronecker, MMSB and B-A baselines respectively (Third row) are shown. Different visualization layouts are used for different datasets.

Comparison with traditional models

Table 1: Comparison of GraphRNNto traditional graph generative models using MMD. $(\max(|V|), \max(|E|))$ of each dataset is shown.

	Community (160,1945)			Ego (399,1071)			Grid (361,684)			Protein (500,1575)		
	Deg.	Clus.	Orbit	Deg.	Clus.	Orbit	Deg.	Clus.	Orbit	Deg.	Clus.	Orbit
E-R	0.021	1.243	0.049	0.508	1.288	0.232	1.011	0.018	0.900	0.145	1.779	1.135
B-A	0.268	0.322	0.047	0.275	0.973	0.095	1.860	0	0.720	1.401	1.706	0.920
Kronecker	0.259	1.685	0.069	0.108	0.975	0.052	1.074	0.008	0.080	0.084	0.441	0.288
MMSB	0.166	1.59	0.054	0.304	0.245	0.048	1.881	0.131	1.239	0.236	0.495	0.775
GraphRNN-S	0.055	0.016	0.041	0.090	**0.006**	0.043	0.029	10^{-5}	0.011	0.057	**0.102**	**0.037**
GraphRNN	**0.014**	**0.002**	**0.039**	**0.077**	0.316	**0.030**	$\mathbf{10^{-5}}$	**0**	$\mathbf{10^{-4}}$	**0.034**	0.935	0.217

- GraphRNN had **80% decrease of MMD** on average compared with traditional baselines
- GraphRNN-S performed well on Protein: may not involve highly complex edge dependencies

Comparison with Deep Learning Models & Generalization

Table 2: GraphRNNcompared to state-of-the-art deep graph generative models on small graph datasets using MMD and negative log-likelihood (NLL). $(\max(|V|), \max(|E|))$ of each dataset is shown. (DeepVAE and GraphVAE cannot scale to the graphs in Table 1.)

	Community-small (20,83)					Ego-small (18,69)				
	Degree	Clustering	Orbit	Train NLL	Test NLL	Degree	Clustering	Orbit	Train NLL	Test NLL
GraphVAE	0.35	0.98	0.54	13.55	25.48	0.13	0.17	0.05	12.45	14.28
DeepGMG	0.22	0.95	0.40	106.09	112.19	0.04	0.10	0.02	21.17	22.40
GraphRNN-S	**0.02**	0.15	**0.01**	31.24	35.94	0.002	**0.05**	**0.0009**	8.51	9.88
GraphRNN	0.03	**0.03**	**0.01**	28.95	35.10	**0.0003**	**0.05**	**0.0009**	9.05	10.61

- GraphRNN had **90% decrease of MMD** on average compared with deep learning baselines
- 22% smaller average NLL gap compared to other deep models

Experiments: Evaluation with Graph Statistics

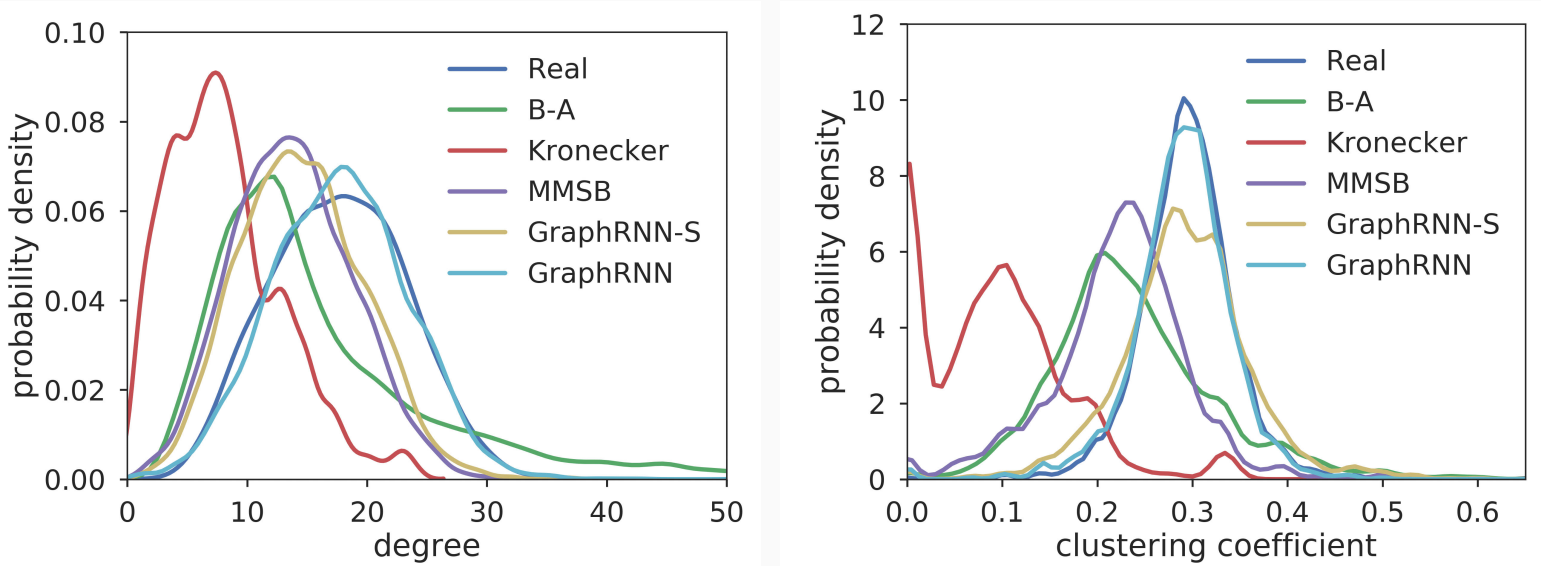


Figure 3: Average degree (Left) and clustering coefficient (Right) distributions of graphs from test set and graphs generated by GraphRNN and baseline models.

- GraphRNN generated graphs' average statistics closely matchs the overall test set distribution.

Experiments: Robustness

Interpolate between (B-A) and (E-R) graphs

Randomly perturb $[0\%, 20\%, ..., 100\%]$ edges of B-A graphs

0% (B-A) $\longleftrightarrow$ 100% (E-R)

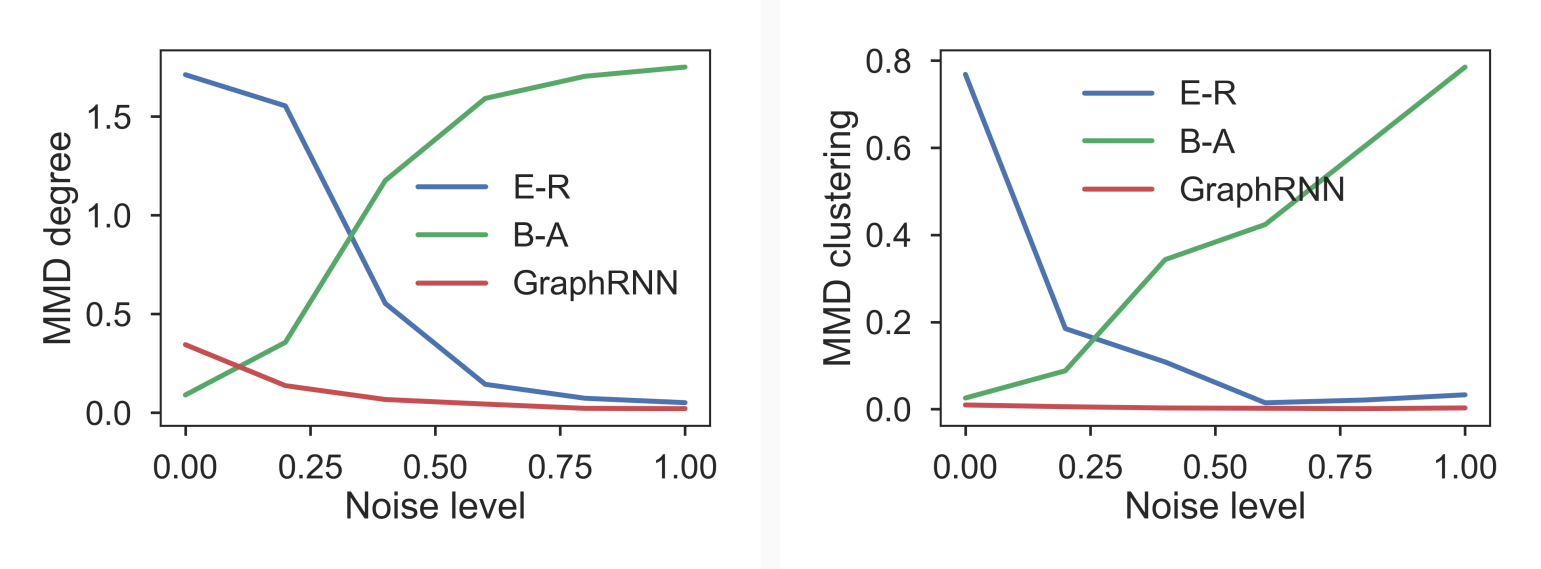


Figure 4: MMD performance of different approaches on degree (Left) and clustering coefficient (Right) under different noise level.

GraphRNN maintains strong performance as we interpolate between these structures, indicating high robustness and versatility.