Deep Generative Models for Graphs: Methods & Applications

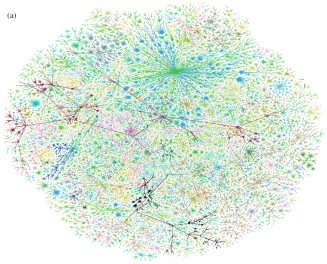
Joint work with **Jiaxuan You**, R. Ying, X. Ren, W. Hamilton, B. Liu, V. Pande

Jure Leskovec

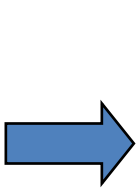
CHAN ZUCKERBERG
BIOHUB

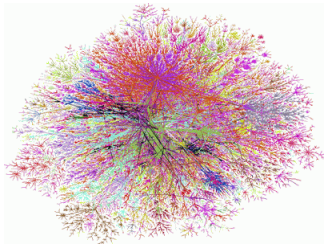
The Problem

- We want to generate realistic graphs

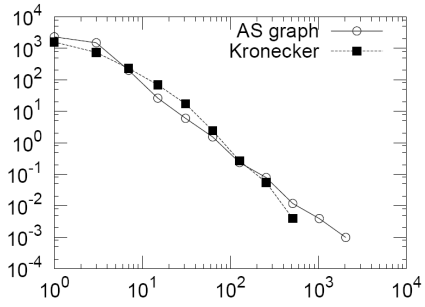


Given a large real graph





Generate a synthetic graph



Some statistical property, e.g., degree distribution

- What is a good model?
- How can we fit the model and generate the graph using it?

Why is This Important?

- Gives insight into the graph formation process
- *Anomaly detection* – abnormal behavior, evolution
- *Predictions* – predicting future from the past
- *Simulations* of novel graph structures
- *Graph completion* – many graphs are partially observed
- "*What if*" scenarios

Graph Generation Tasks

Task 1: Realistic graph generation

- Generate graphs that are similar to a given dataset

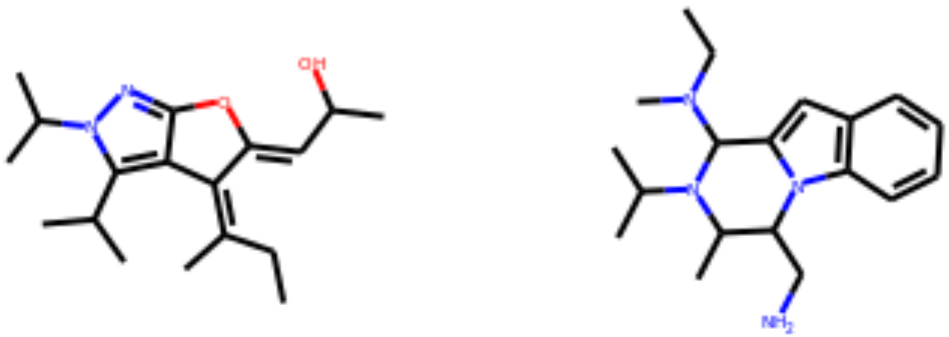
Task 2: Goal-directed graph generation

- Generate graphs that optimize given objectives/constraints
  - Drug molecule generation/optimization

Why is it Interesting?

Drug discovery

- Discover highly drug-like molecules

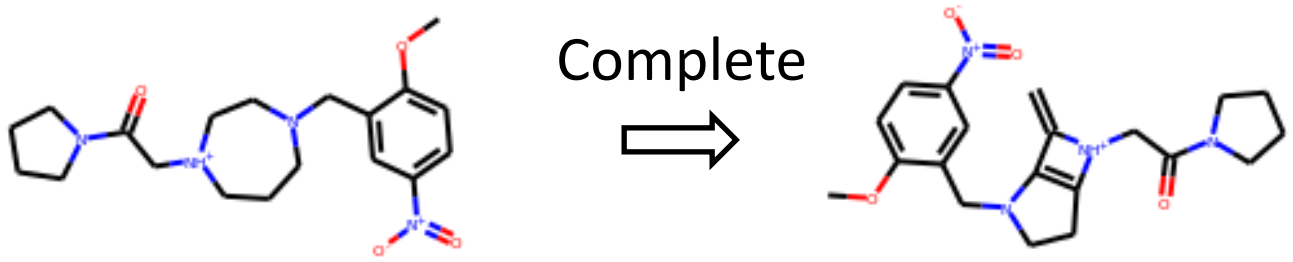


`drug_likeness=0.94`

Why is it Interesting?

Drug discovery

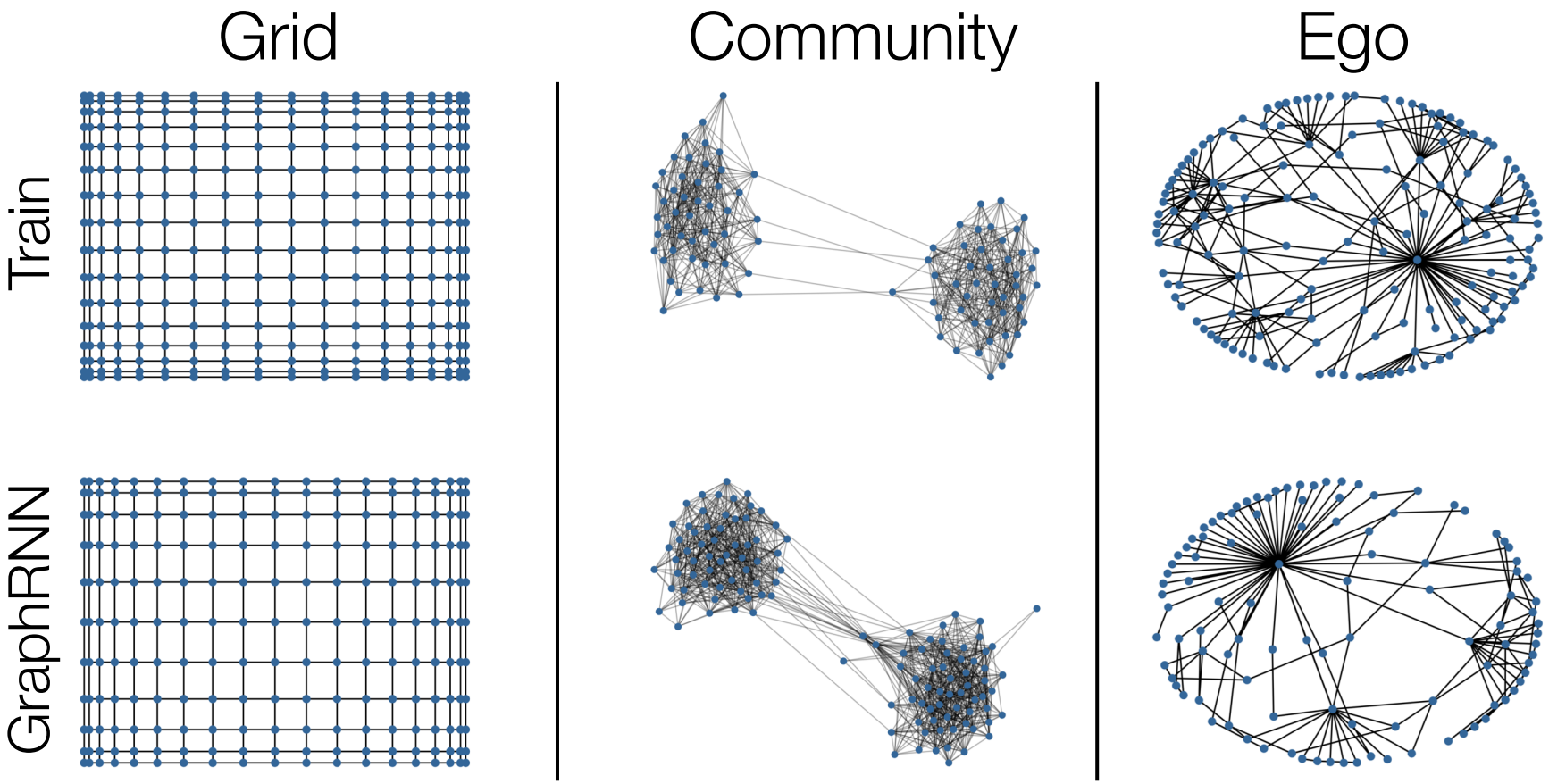
- Complete an existing molecule to have a desired property



Solubility=-5.55 ⟹ (Improve) Solubility=-1.78

Why is it Interesting?

Discovering novel structures

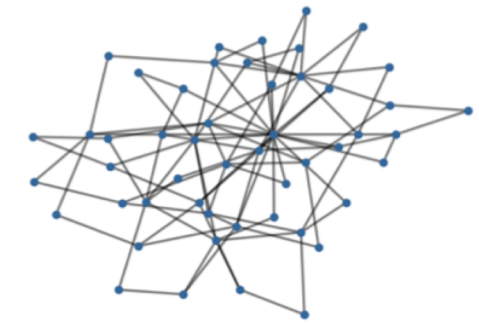


Why is it Interesting?

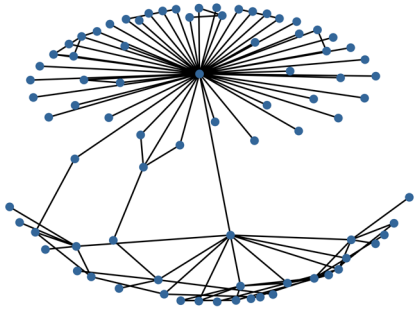
Network Science

- Null models for realistic networks

```
Barabasi_Albert(n=50, m=2) ~
```



```
NeuralNet_X(n=50, p=3, q=5, …) ~
```



Why is it Hard?

- Large and variable output space
  - For n nodes we need to generate n^2 values
  - Graph size (nodes, edges) varies
- Non-unique representations:
  - n-node graph can be represented in $n!$ Ways
  - Hard to compute/optimize objective functions (e.g., reconstruction error)
    - GraphVAE solves approx. graph matching, $O(n^4)$
- Complex dependencies:
  - Edge formation has long-range dependencies

Key Insight

Generating graphs via sequentially adding nodes/edges

Benefits:

- Represents graphs with different sizes with different sequence lengths
- Corresponds different node orderings to different generation trajectories
- Captures complex dependencies between nodes, e.g., the triad closure property

Key Insights

- **Sequential generation of nodes/edges is** backbone of many existing works:
  - Learning Deep Generative Models of Graphs
    Y. Li et al., ICML 2018.
  - Junction Tree Variational Autoencoder for Molecular Graph Generation
    W. Jin et al., ICML 2018.
  - GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models
    J. You et al., ICML 2018.
  - Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation
    J. You et al., NeurIPS 2018.
- Other related works
  - Variational Graph Auto-Encoders
    T. Kipf, M. Welling, NeurIPS 2016 Workshop.
  - NetGAN: Generating Graphs via Random Walks
    A. Bojchevski et al., ICML 2018.
  - GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders
    M. Simonovsky, N. Komodakis, arXiv.
  - Graphite: Iterative Generative Modeling of Graphs
    A. Grover et al., arXiv.

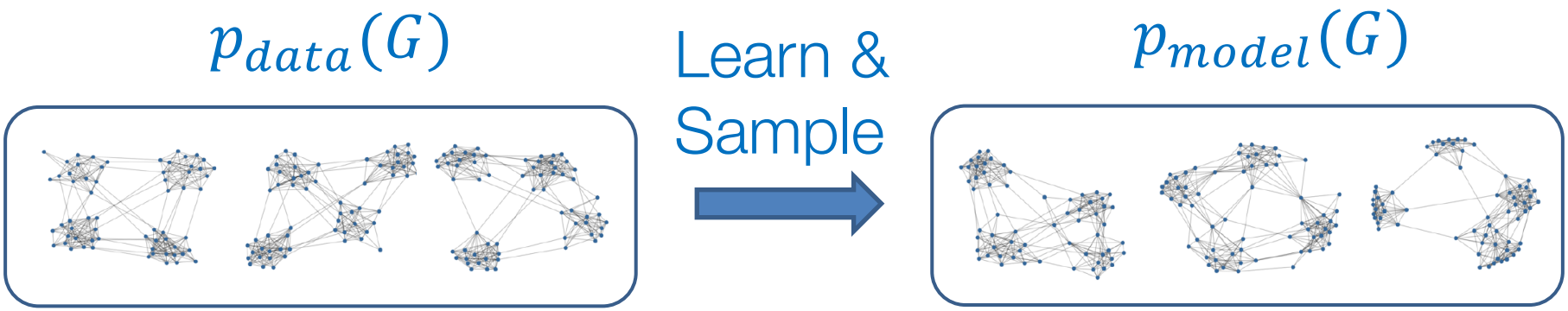
GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models

GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models.
J. You, R. Ying, X. Ren, W. Hamilton, J. Leskovec. *ICML*, 2018.

Data & Code: https://github.com/snap-stanford/GraphRNN

Graph Generative Model

- **Given:** Graphs from $p_{data}(G)$
- **Goal:**
  - Learn the distribution $p_{model}(G)$
  - Sample from $p_{model}(G)$



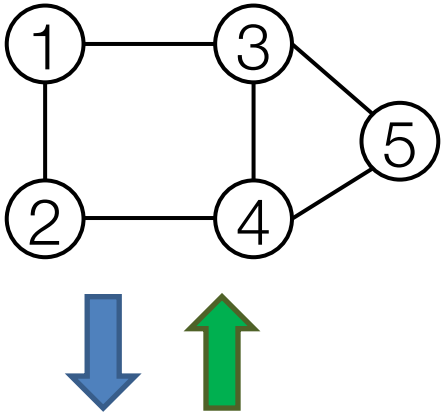
Graph Generative Model

- Challenges for modeling $p_{data}(G)$:
  - Variable graph size
  - Numerous node orderings
  - Complex node dependency
- **Solution:** Map graphs to a different representation that is easier to learn and sample from

Model Graphs as Sequences

Graph G with node ordering π can be uniquely mapped into a sequence of node and edge additions S^{π}

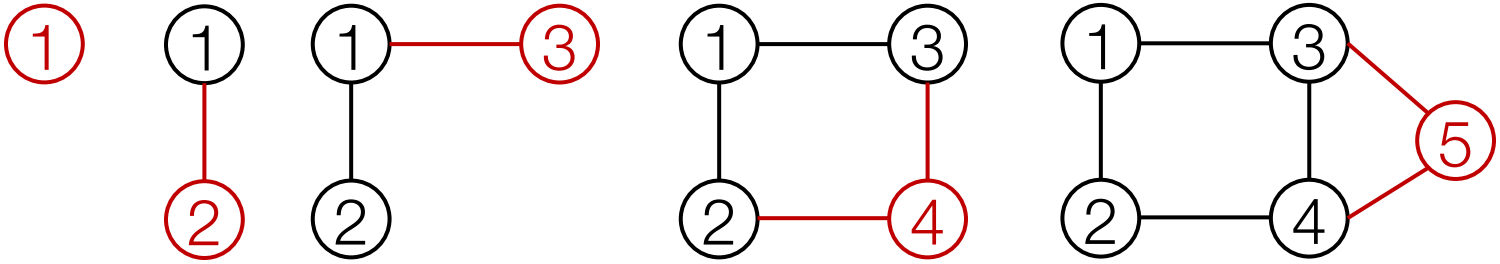
Graph G with node ordering π:



Sequence S^{π}:

S_4^{π} = ("Connect 4 and 2", "Connect 4 and 3")

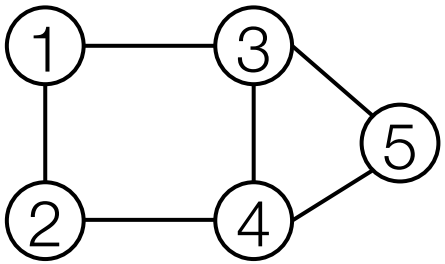
S^{π} is a seq. of seq.!



$$S^{\pi} = (S_1^{\pi} , S_2^{\pi} , S_3^{\pi} , S_4^{\pi} , S_5^{\pi})$$

Model Graphs as Sequences

A Graph G with node ordering π can be uniquely represented as a sequence S^{π}



Graph G

Map graph to a sequence

$S^{\pi} = f_S(G, \pi)$

Map sequence to a graph

$G = f_G(S^{\pi})$

S_1^{π}	S_2^{π}	S_3^{π}	S_4^{π}	S_5^{π}
0	1	1	0	0
1	0	0	1	0
1	0	0	1	1
0	1	1	0	1
0	0	1	1	0

Sequence S^{π}

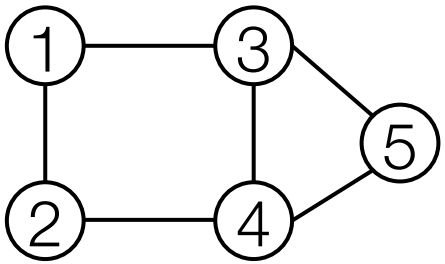
S_i^{π} are blue columns

Model Graphs as Sequences

Likelihood of graph ➔ Likelihood of sequence

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

Model the likelihood of seq. with an RNN:



Graph G

Map graph to a sequence

$S^\pi = f_S(G, \pi)$

Map sequence to a graph

$G = f_G(S^\pi)$

S_1^π	S_2^π	S_3^π	S_4^π	S_5^π
0	1	1	0	0
1	0	0	1	0
1	0	0	1	1
0	1	1	0	1
0	0	1	1	0

Sequence S^π

S_i^π are blue columns

GraphRNN: Two levels of RNN

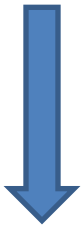
- **Goal:** Model graph generation as a sequence generation
- **Need to model two processes:**
  - Generate a state for a new node (Node-level RNN)
  - Generate edges for the new node based on its state (Edge-level RNN)

Sequences Generation

Node-level RNN 

Generate state for a new node

S_1^π	S_2^π	S_3^π	S_4^π	S_5^π
	1	1	0	0
		0	1	0
			1	1
				1



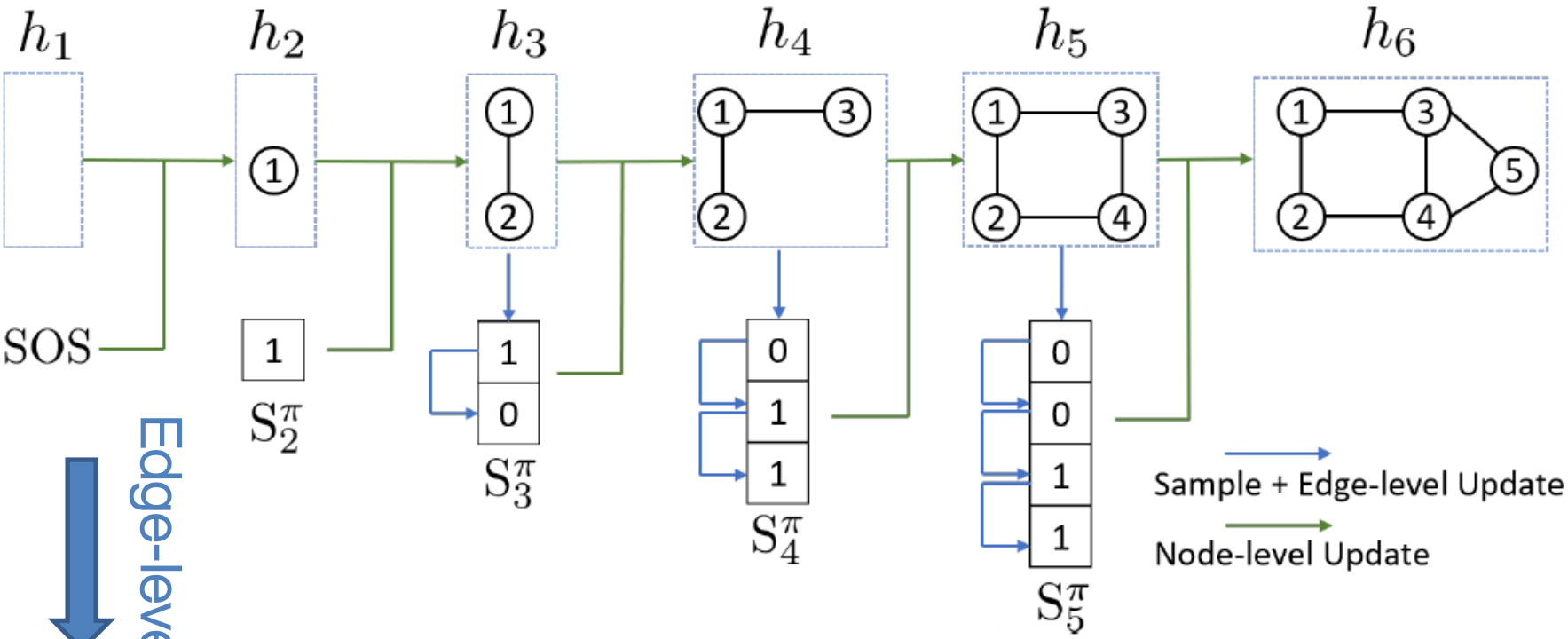
Edge-level RNN

Generate edges for the new node (based on the state of Node RNN)

GraphRNN Architecture

$$p(S^\pi) = \prod_{i=1}^{n+1} p(S_i^\pi | S_1^\pi, ..., S_{i-1}^\pi)$$

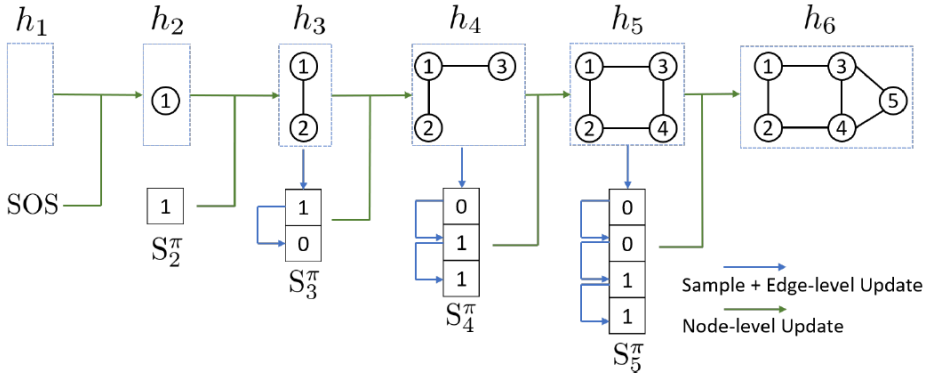
Node-level RNN



Edge-level RNN

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

GraphRNN Architecture



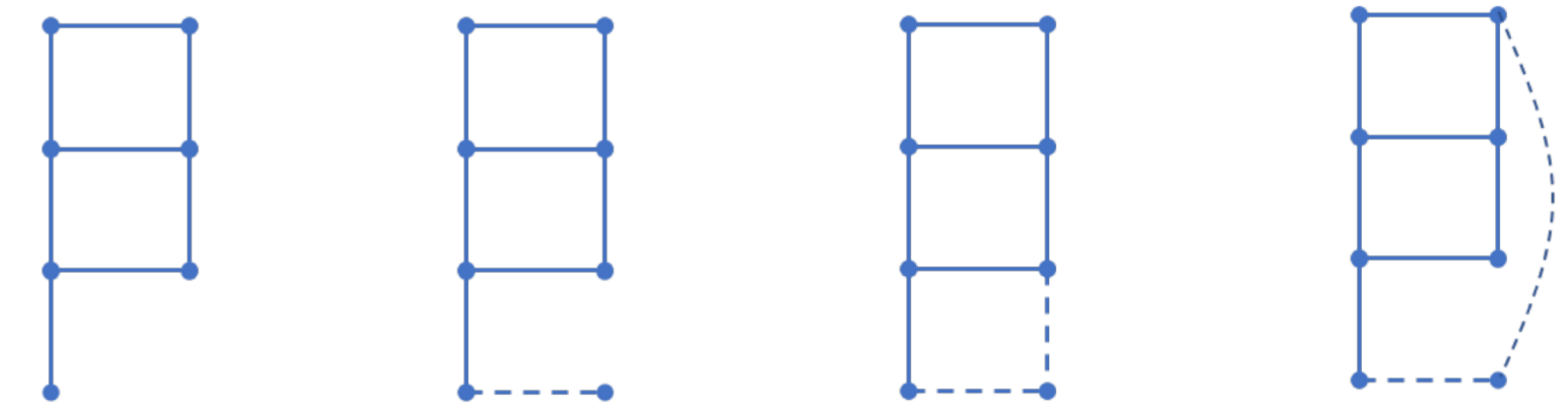
- Node-level RNN: GRU
- Edge-level RNN:

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

Dependent Bernoulli sequence:

- Edge RNN initialized by Node RNN
- Output mapped to an edge prob., flip coin
- Feed back the coin flip to the Edge RNN

Example: Generating a Ladder

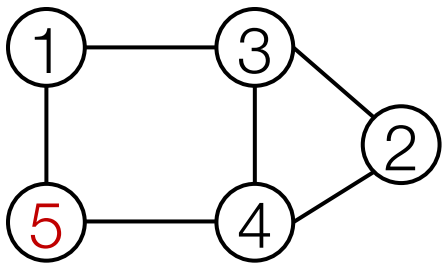


To generate a ladder the model needs to capture dependencies

GraphRNN does this via the Edge-level RNN

Issue: Tractability

- Any node can connect to any prior node
- Too many steps for edge generation
  - Need to generate full adjacency matrix
  - Complex too-long edge dependencies



"Recipe" to generate the left graph:
- Add node 1
- Add node 2
- Add node 3
- Connect 3 with 1 and 2
- Add node 4
- …

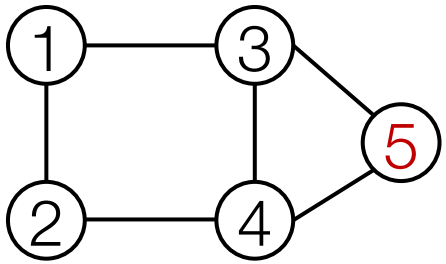
Random node ordering:

Node 5 may connect to any/all previous nodes

How do we limit complexity of graph generation?

Solution: Tractability via BFS

- **Breadth-First Search node ordering**



BFS ordering

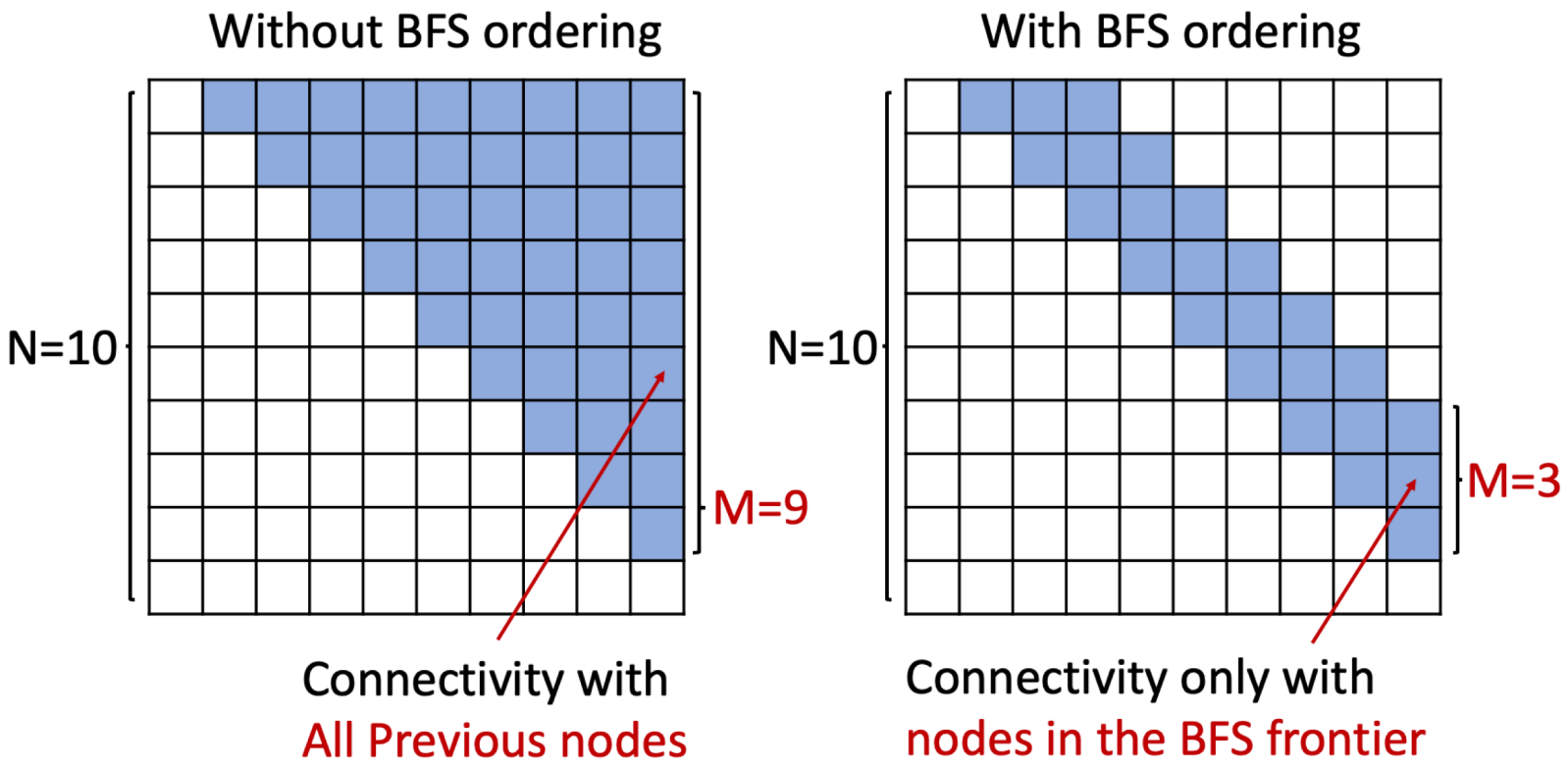
"Recipe" to generate the left graph:
- Add node 1
- Add node 2
- Connect 2 with 1
- Add node 3
- Connect 3 with 1
- Add node 4
- Connect 4 with 2 and 3

BFS node ordering: Node 5 will never connect to node 1 (only need memory of 2 "steps" rather than $n - 1$ steps)

- **Benefits:**
  - Reduce possible node orderings
  - Reduce steps for edge generation

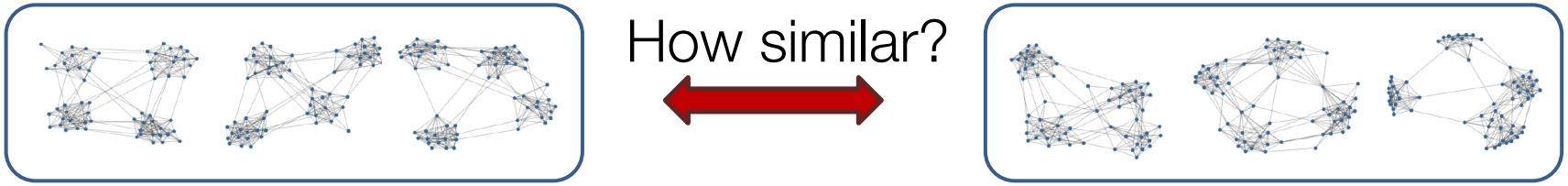
Tractability via BFS

BFS reduces the number of steps for edge generation



Quantitative Comparison

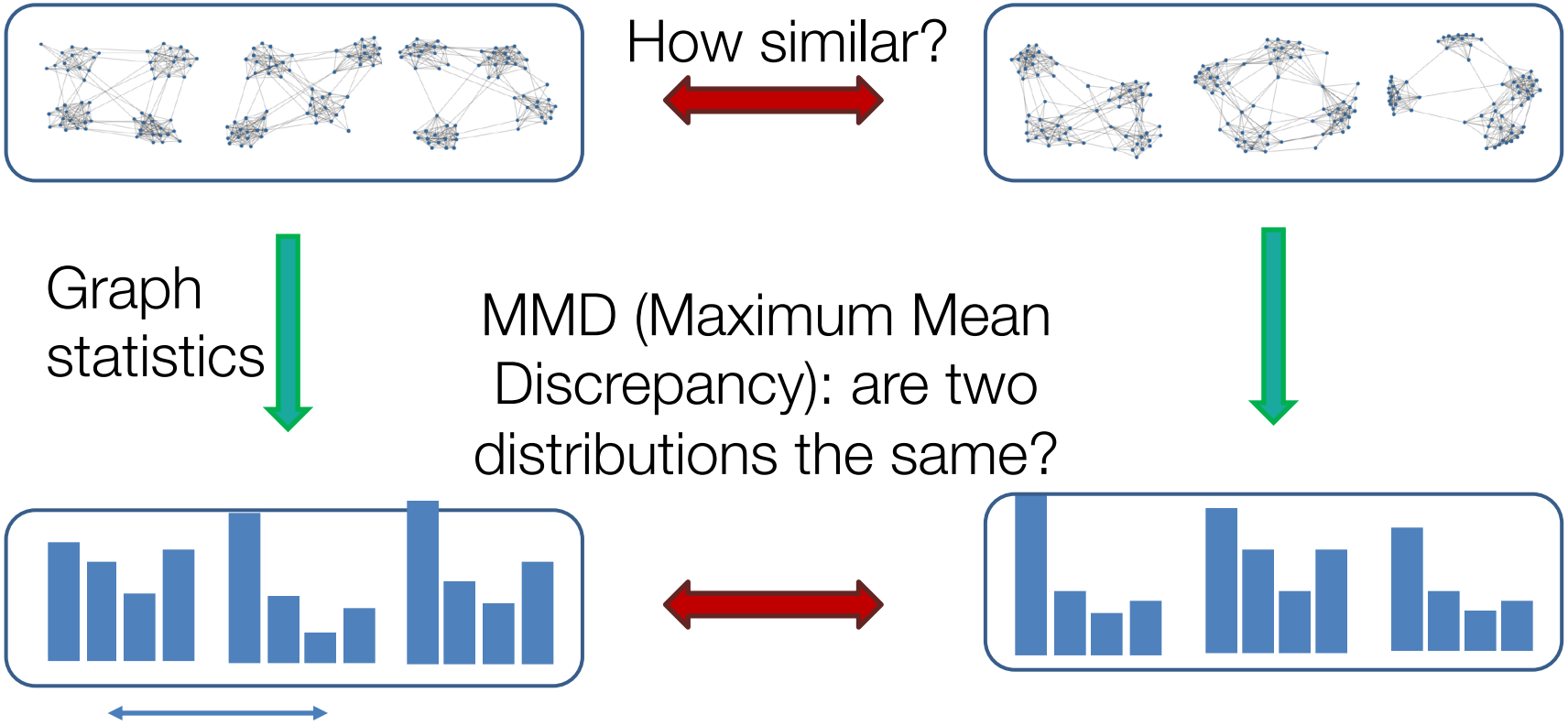
Task: Compare two sets of graphs



- **Issue:** Want to compare a set of training and generated graphs
- **Solution:**
  - Compare two graph statistics
    - Earth Mover Distance (EMD)
  - Compare sets of graph statistics
    - Maximum Mean Discrepancy (MMD) based on EMD

Quantitative Comparison

Task: Compare two sets of graphs



Baselines

- Classical graph generative models
  - Barabasi-Albert (B-A) [Barabasi&Albert, 1998]
  - Erdos-Renyi model [Erdos&Renyi, 1959]
- Model with learnable parameters
  - Kronecker graphs [Leskovec et al., 2010]
  - Mixed-membership Stochastic Block model [Airoldi et al., 2008]
- Recent deep models (can't scale beyond ~30 nodes)
  - GraphVAE [M. Simonovsky, N. Komodakis et al., 2017]
  - DeepGMG [Y. Li et al. 2017]

Quantitative Comparison

	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

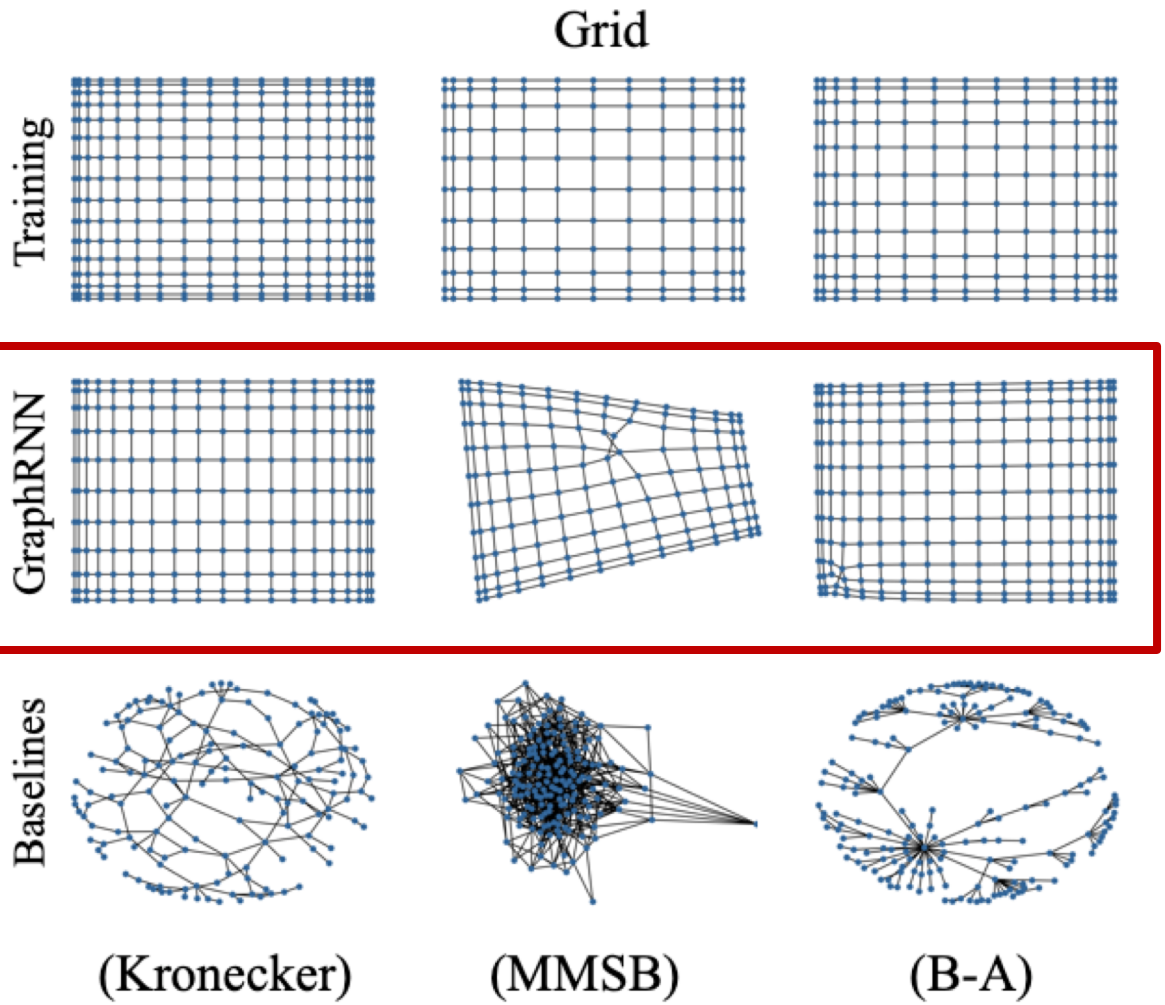
MMD score. Other DL methods don't scale to this data

	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

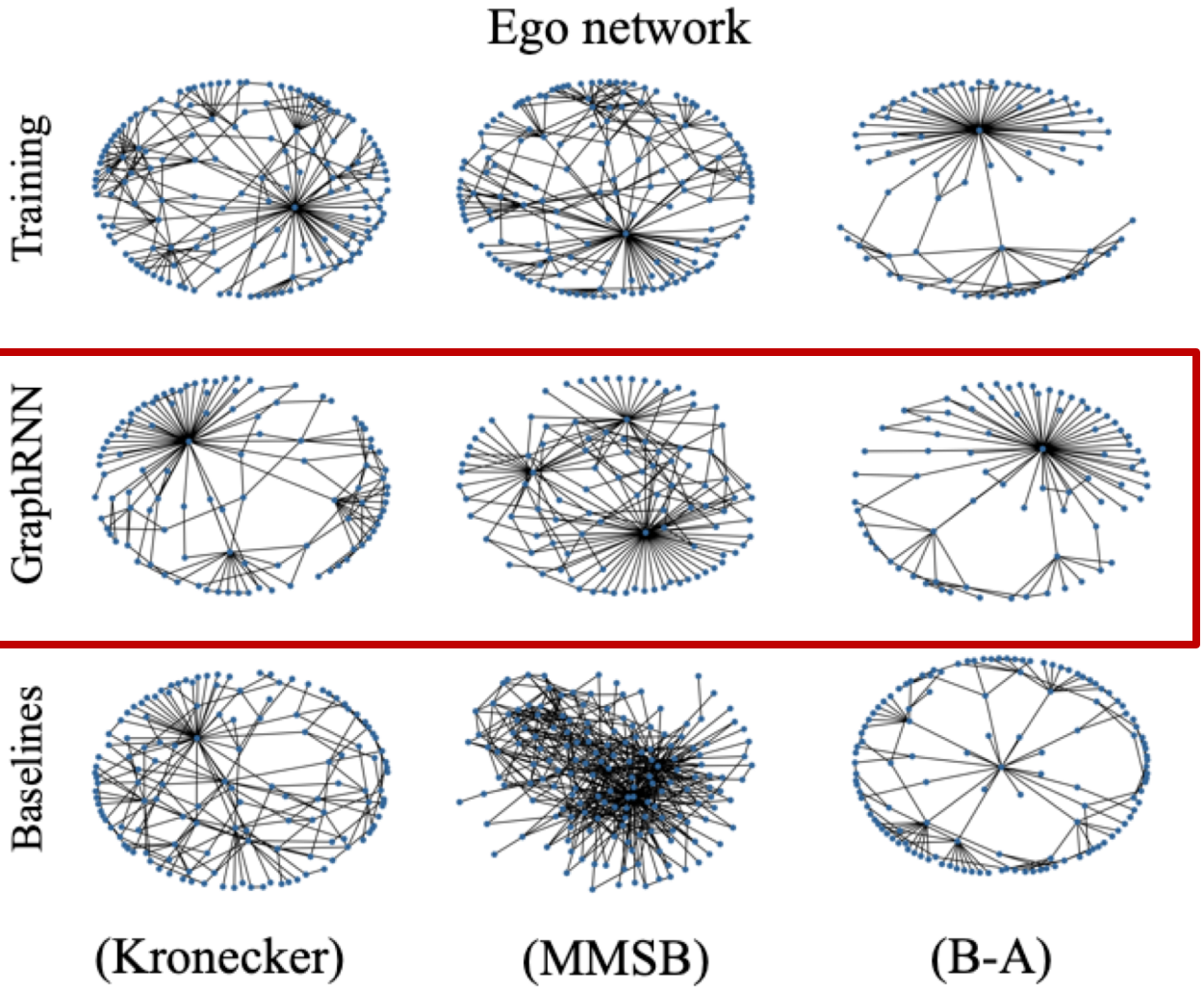
GraphRNN achieves best performance:

- 80% improvement vs. traditional baselines
- 90% improvement vs. DL baselines

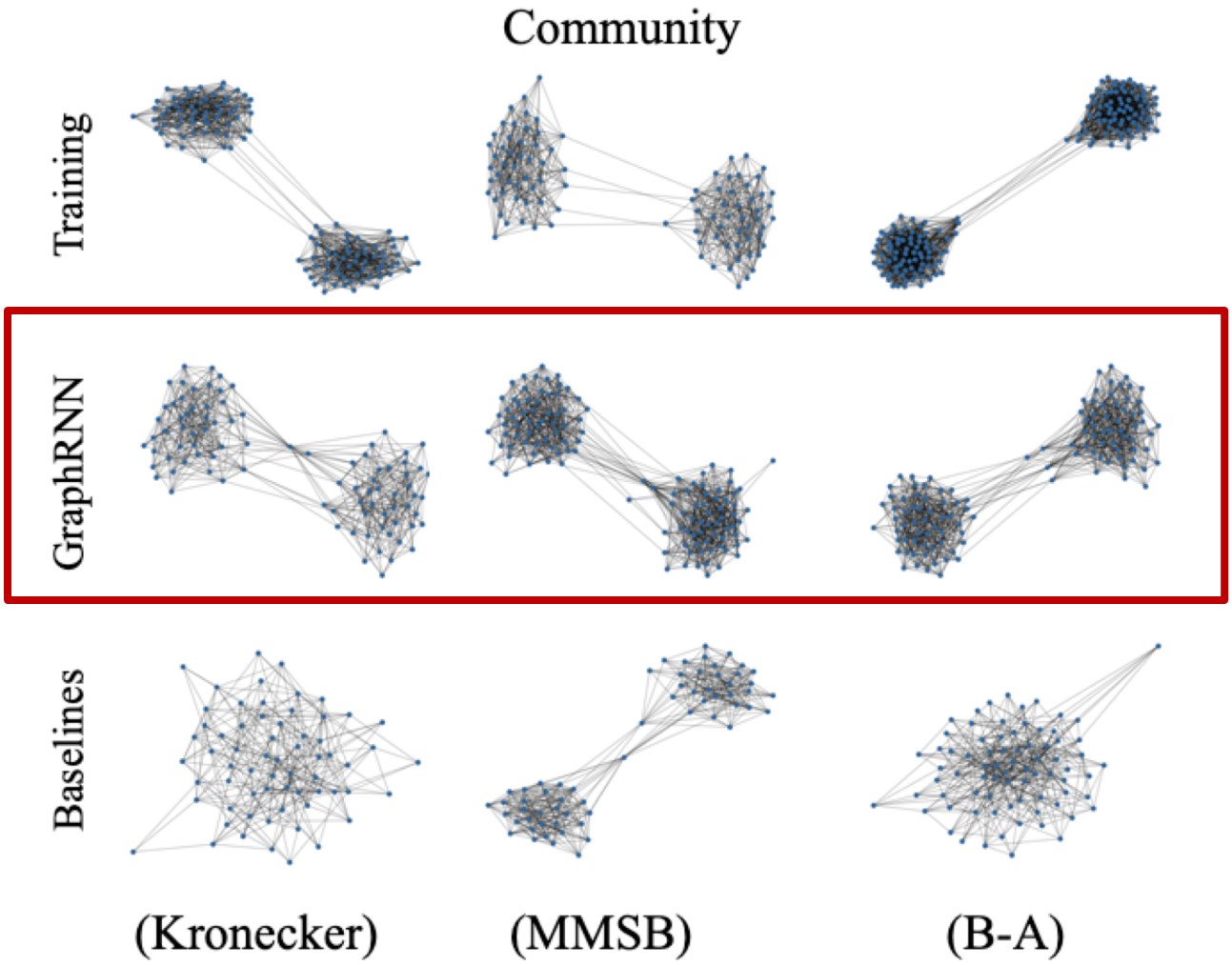
Qualitative Comparison



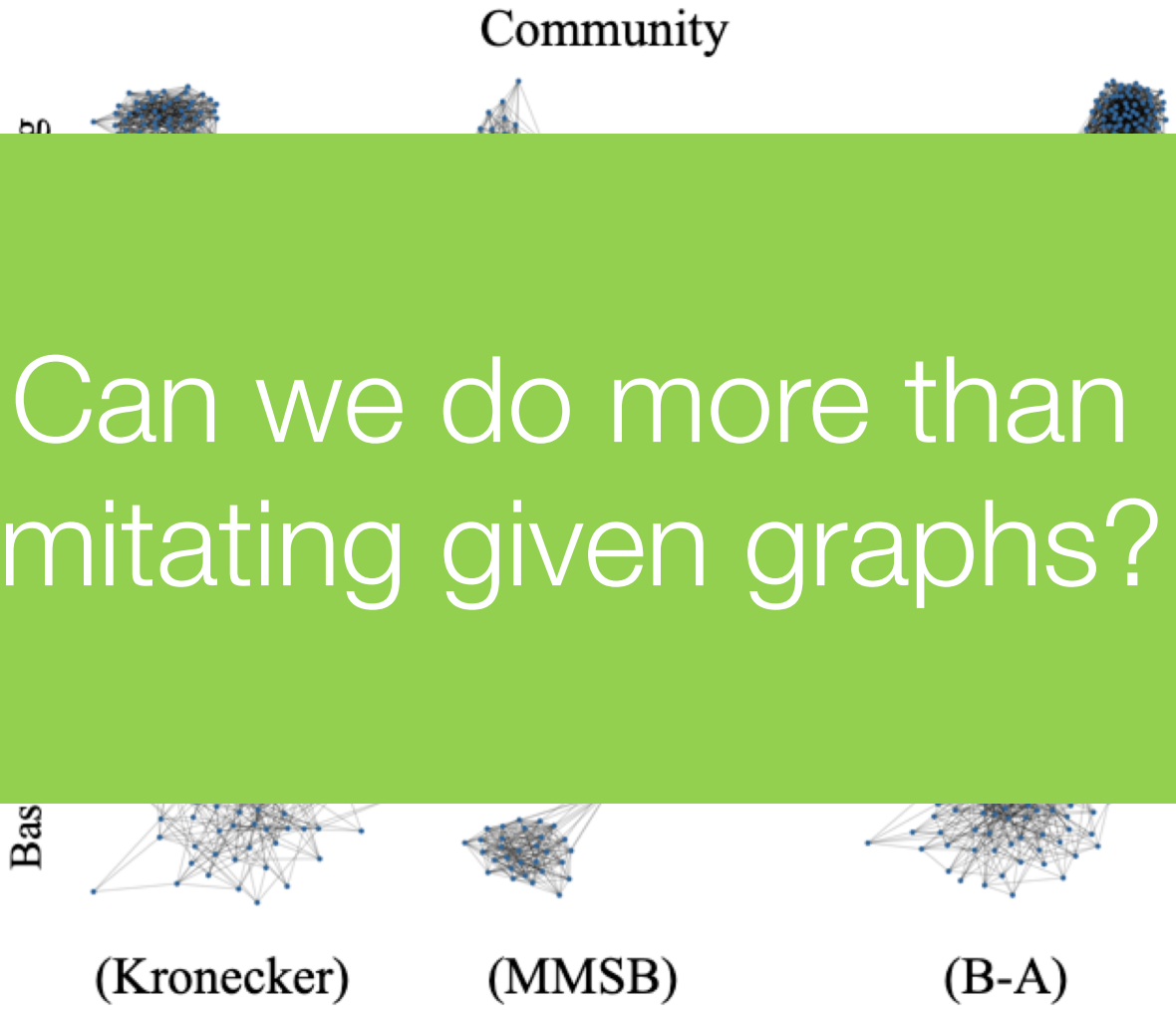
Qualitative Comparison



Qualitative Comparison



Qualitative Comparison



Can we do more than imitating given graphs?

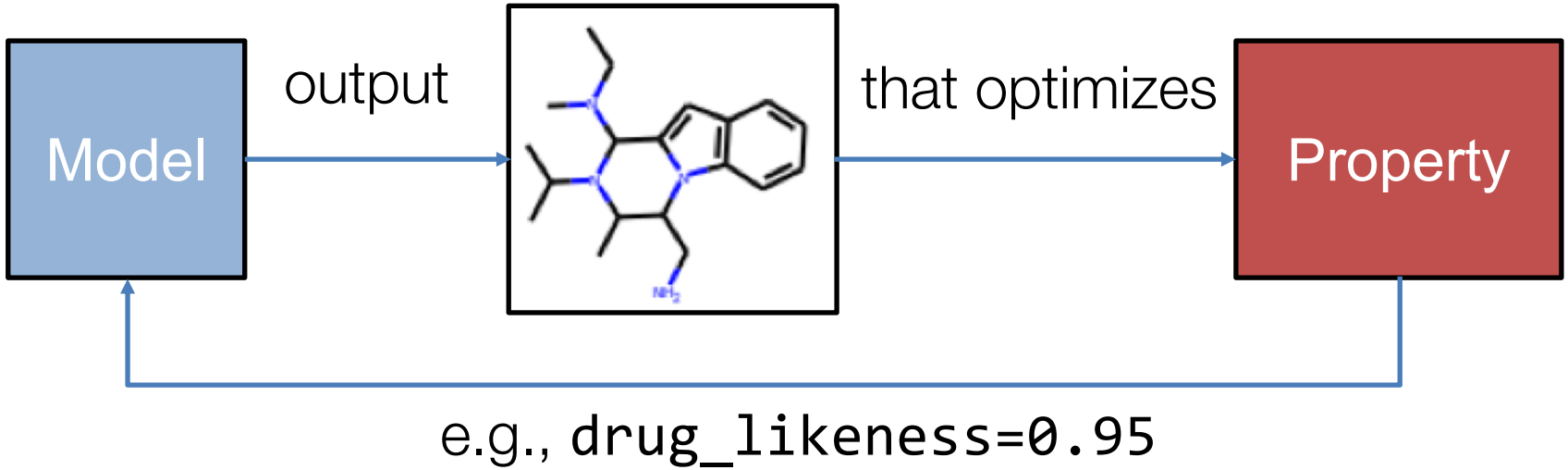
Graph Convolutional Policy Network: Goal-Directed Graph Generation

Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation.
J. You, B. Liu, R. Ying, V. Pande, J. Leskovec. *NeurIPS*, 2018.

Data & code: https://github.com/bowenliu16/rl_graph_generation

Motivation

Question: Can we learn a model that can generate **valid** and **realistic** molecules with **high value of a given chemical property**?



Goal-Directed Graph Gen.

Generating graphs that:

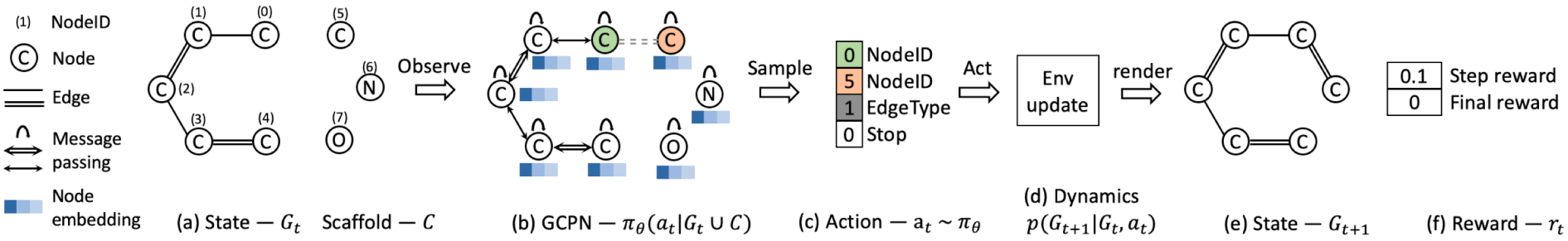
- Optimize a given objective (High scores)
  - e.g., drug-likeness (black box)
- Obey underlying rules (Valid)
  - e.g., chemical valency
- Are learned from examples (Realistic)
  - e.g., Imitating a molecule graph dataset

Graph Conv. Policy Network

Graph Convolutional Policy Network combines graph representation + RL:

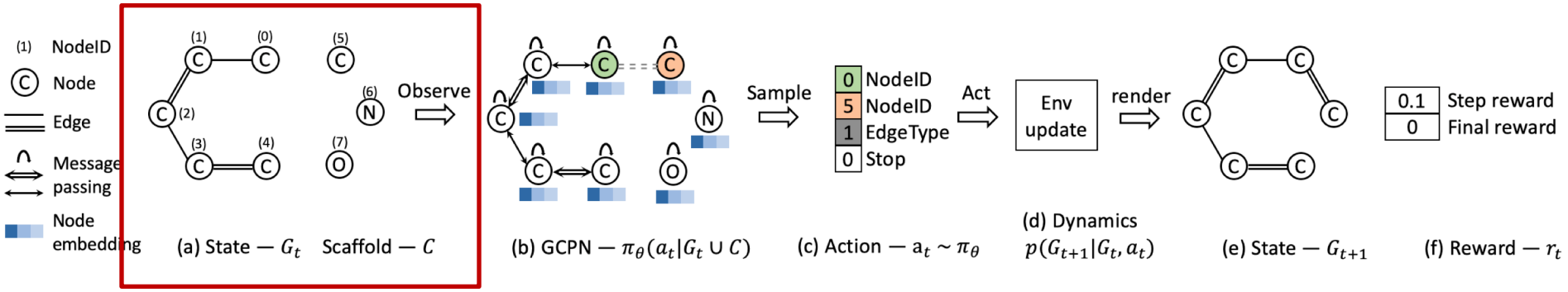
- Graph representation captures complex structural information, and enables validity check in each state transition (Valid)
- Reinforcement learning optimizes intermediate/final rewards (High scores)
- Adversarial training imitates examples in given datasets (Realistic)

Overview of GCPN



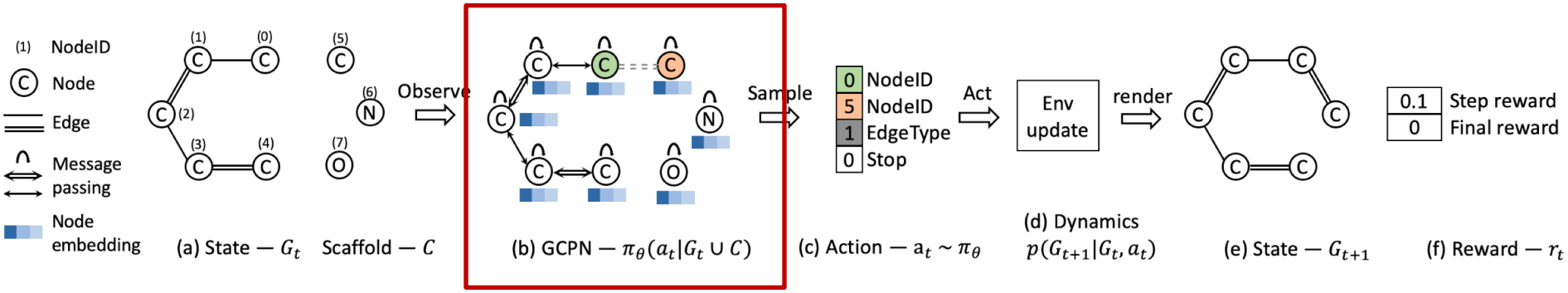
- (a) Insert nodes/scaffolds
- (b) Compute state via GCN
- (c) Saple next action
- (d) Take action (check chemical validity)
- (e, f) Compute reward

GCPN Framework



Step(a) Initialization: Insert possible new nodes C, along with a partially generated graph G_t

GCPN Framework

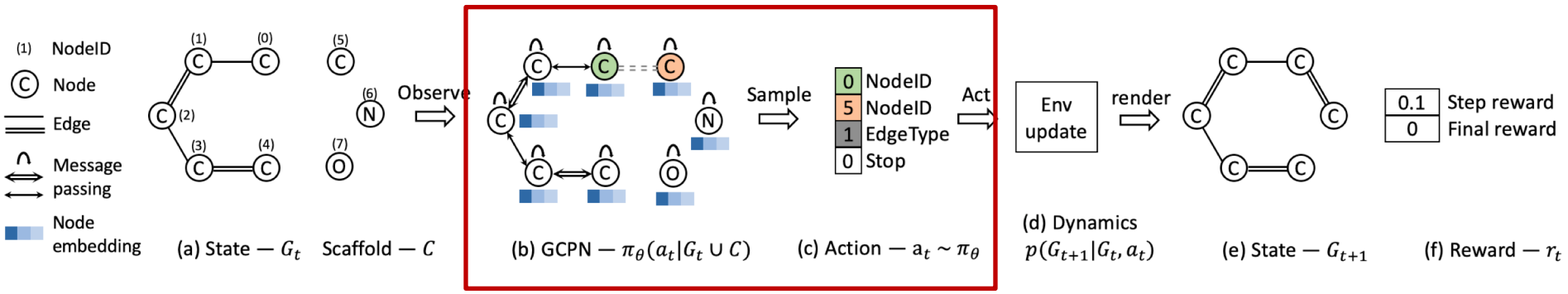


Step(b) Policy: First Compute node embeddings for all the nodes via GCN

$$H^{(l+1)} = \text{AGG}(\text{ReLU}(\{\tilde{D}_i^{-\frac{1}{2}}\tilde{E}_i\tilde{D}_i^{-\frac{1}{2}}H^{(l)}W_i^{(l)}\}, \forall i \in (1,...,b)))$$

Semi-supervised classification with graph convolutional networks T. Kipf, M. Welling, ICLR 2016

GCPN Framework



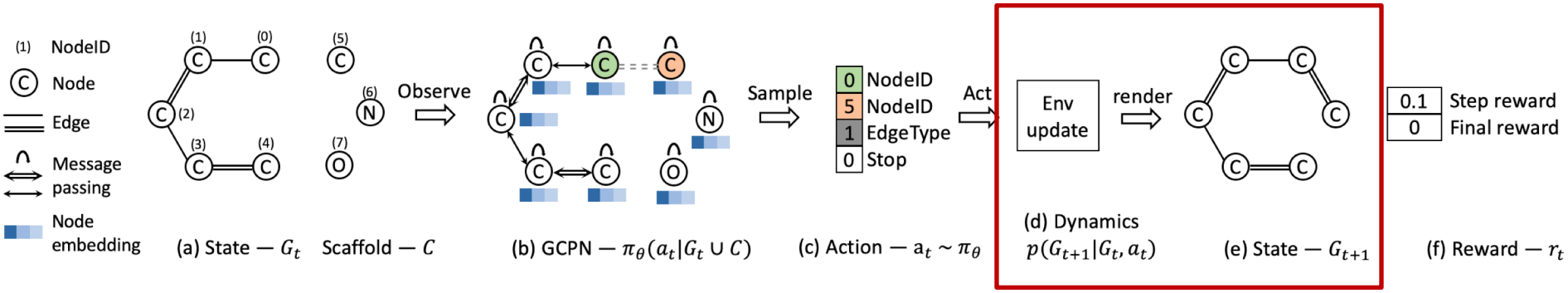
Step(c) Policy: Compute policy π_θ based on node embeddings, sample action a_t

$$a_t = \text{CONCAT}(a_{\text{first}}, a_{\text{second}}, a_{\text{edge}}, a_{\text{stop}})$$

$$f_{\text{first}}(s_t) = \text{SOFTMAX}(m_f(X)),$$
$$f_{\text{second}}(s_t) = \text{SOFTMAX}(m_s(X_{a_{\text{first}}}, X)),$$
$$f_{\text{edge}}(s_t) = \text{SOFTMAX}(m_e(X_{a_{\text{first}}}, X_{a_{\text{second}}})),$$
$$f_{\text{stop}}(s_t) = \text{SOFTMAX}(m_t(\text{AGG}(X))),$$

$$a_{\text{first}} \sim f_{\text{first}}(s_t) \in \{0,1\}^n$$
$$a_{\text{second}} \sim f_{\text{second}}(s_t) \in \{0,1\}^{n+c}$$
$$a_{\text{edge}} \sim f_{\text{edge}}(s_t) \in \{0,1\}^b$$
$$a_{\text{stop}} \sim f_{\text{stop}}(s_t) \in \{0,1\}$$

GCPN Framework

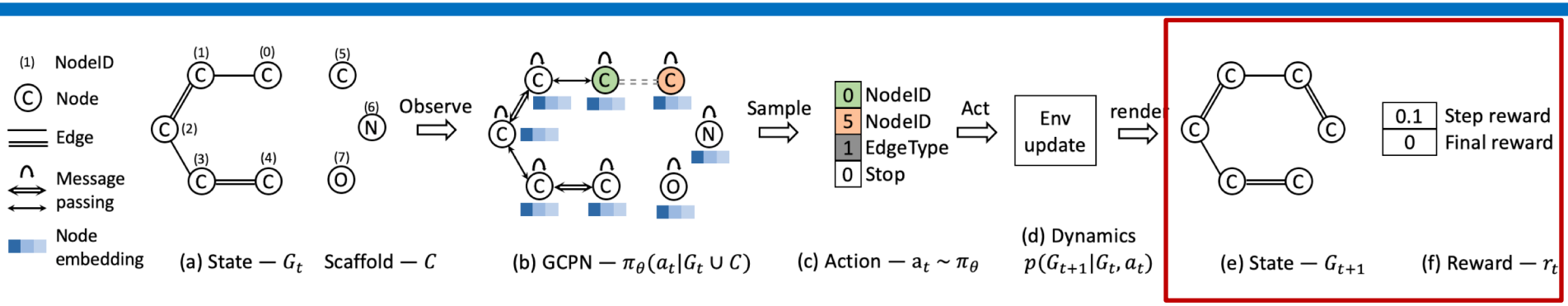


- Step(d) State transition: Check if validity constraints (e.g., chemical valency rules) are met, reject invalid actions, then make a transition $p(G_{t+1}|G_t, a_t)$

How Do We Set the Reward?

- Learn to take valid action
  - At each step, assign small positive reward for valid action
- Optimize desired properties
  - At the end, assign positive reward for high desired property
- Generate realistic graphs
  - At the end, adversarially train a GCN discriminator, compute adversarial rewards that encourage realistic molecule graphs

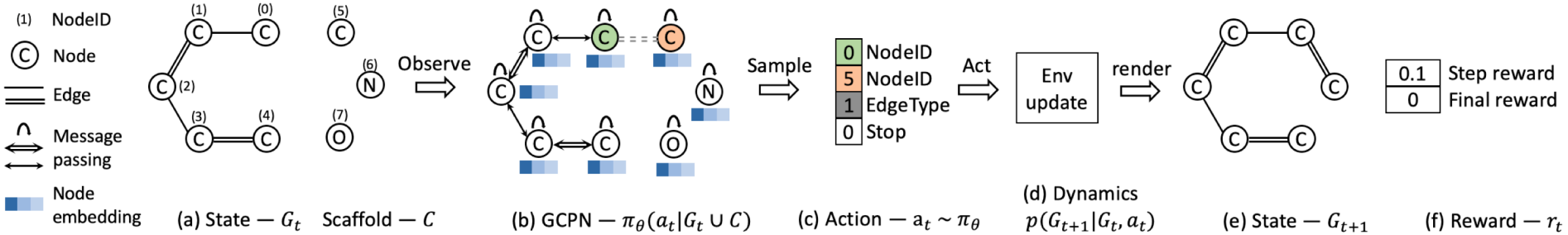
How Do We Set the Reward?



Reward: r_t = Final reward + Step reward

- Final reward = Domain-specific reward + Adversarial reward
- Step rewards = Step-wise validity reward

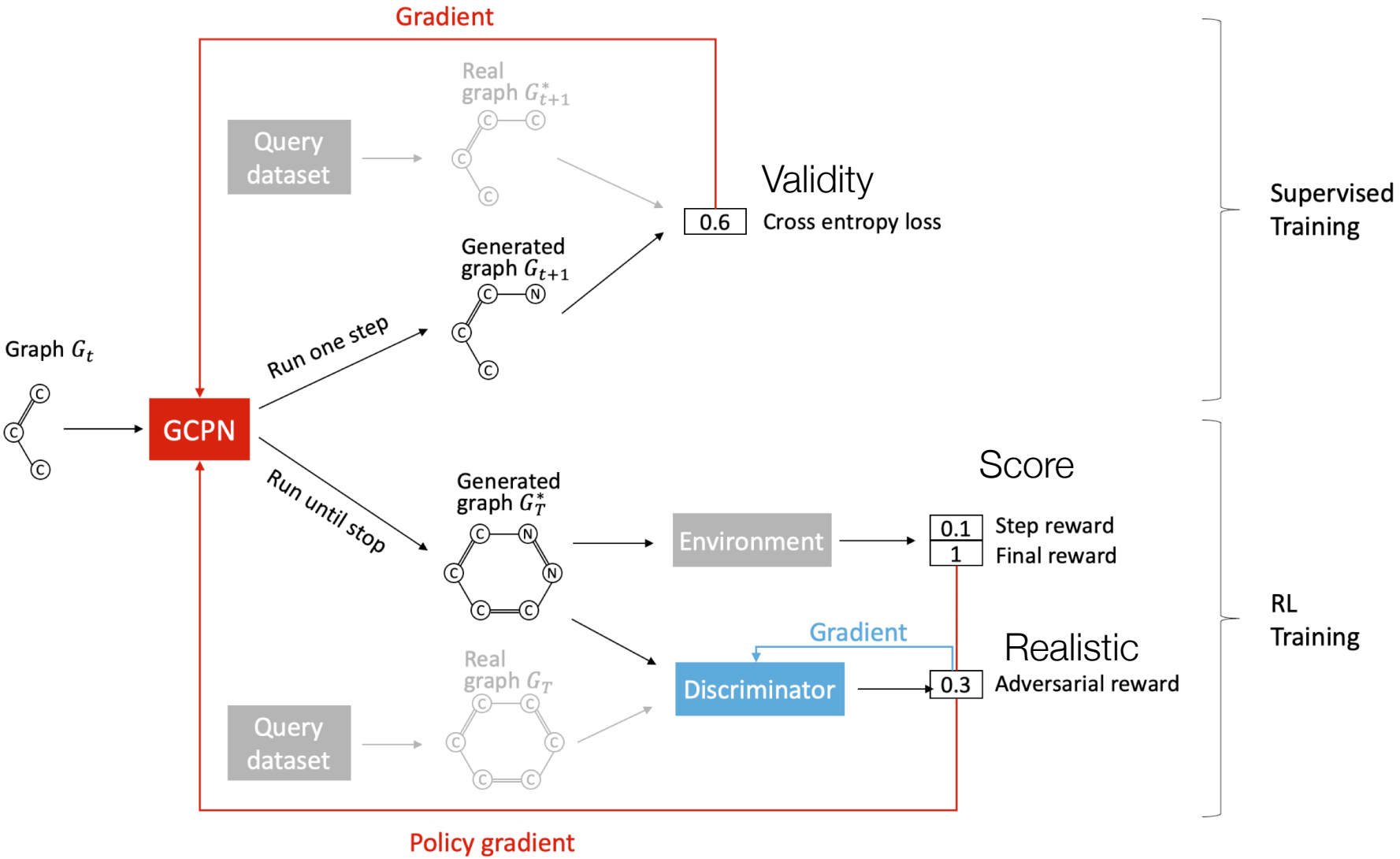
How Do We Train?



- First pretrain policy by imitate generating realistic graphs (Supervised expert pretraining).
- Then train to optimize rewards using PPO policy gradient algorithm.

Proximal policy optimization algorithms J. Schulman et al., arXiv 2017

GCPN Architecture



GCPN: Tasks

- Property optimization
  - Generate molecules with high specified property score
- Property targeting
  - Generate molecules whose specified property score falls within given range
- Constrained property optimization
  - Edit a given molecule for a few steps to achieve higher specified property score

Data and Baselines

- ZINC250k dataset
  - 250,000 drug like molecules whose maximum atom number is 38
- Baselines:
  - ORGAN: String representation + RL [Guimaraes et al., 2017]
  - JT-VAE: VAE-based vector representation + Bayesian optimization [Jin et al., 2018]

Quantitative Results

Property optimization

- +60% higher property scores

Table 1: Comparison of the top 3 property scores of generated molecules found by each model.

Method	Penalized logP				QED			
	1st	2nd	3rd	Validity	1st	2nd	3rd	Validity
ZINC	4.52	4.30	4.23	100.0%	0.948	0.948	0.948	100.0%
ORGAN	3.63	3.49	3.44	0.4%	0.896	0.824	0.820	2.2%
JT-VAE	5.30	4.93	4.49	100.0%	0.925	0.911	0.910	100.0%
GCPN	**7.98**	**7.85**	**7.80**	**100.0%**	**0.948**	**0.947**	**0.946**	**100.0%**

logP: octanol-water partition coef., indicates solubility
QED: indicator of drug-likeness

Quantitative Results

Property targeting

- 7x higher success rate than JT-VAE, 10% less diversity

Table 2: Comparison of the effectiveness of property targeting task.

Method	$-2.5 \leq \text{logP} \leq -2$		$5 \leq \text{logP} \leq 5.5$		$150 \leq \text{MW} \leq 200$		$500 \leq \text{MW} \leq 550$	
	Success	Diversity	Success	Diversity	Success	Diversity	Success	Diversity
ZINC	0.3%	0.919	1.3%	0.909	1.7%	0.938	0	–
JT-VAE	11.3%	**0.846**	7.6%	0.907	0.7%	0.824	16.0%	0.898
ORGAN	0	–	0.2%	**0.909**	15.1%	0.759	0.1%	0.907
GCPN	**85.5%**	0.392	**54.7%**	0.855	**76.1%**	**0.921**	**74.1%**	**0.920**

logP: octanol-water partition coef., indicates solubility
MW: molecular weight an indicator of drug-likeness
Diversity: avg. pairwise Tanimoto distance between Morgan fingerprints of molecules

Quantitative Results

Constrained property optimization

- +180% higher scores than JT-VAE

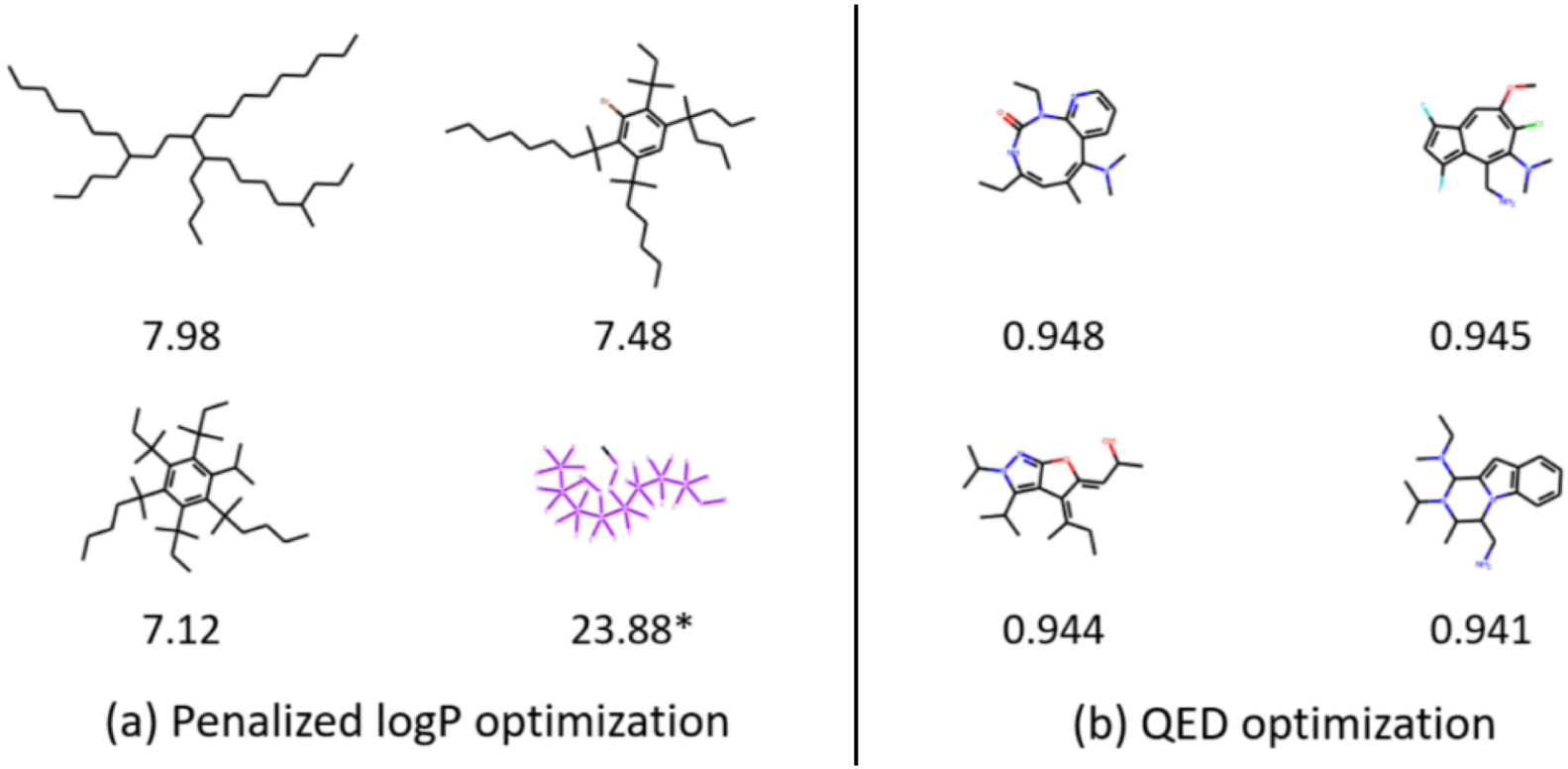
Table 3: Comparison of the performance in the constrained optimization task.

δ	JT-VAE			GCPN		
	Improvement	Similarity	Success	Improvement	Similarity	Success
0.0	1.91 ± 2.04	0.28 ± 0.15	97.5%	$\mathbf{4.20 \pm 1.28}$	$\mathbf{0.32 \pm 0.12}$	**100.0%**
0.2	1.68 ± 1.85	0.33 ± 0.13	97.1%	$\mathbf{4.12 \pm 1.19}$	$\mathbf{0.34 \pm 0.11}$	**100.0%**
0.4	0.84 ± 1.45	$\mathbf{0.51 \pm 0.10}$	83.6%	$\mathbf{2.49 \pm 1.30}$	0.47 ± 0.08	**100.0%**
0.6	0.21 ± 0.71	0.69 ± 0.06	46.4%	$\mathbf{0.79 \pm 0.63}$	$\mathbf{0.68 \pm 0.08}$	**100.0%**

Qualitative Results

Visualization of GCPN graphs:

Property optimization

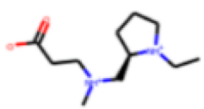


Qualitative Results

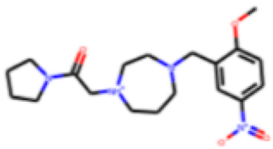
Visualization of GCPN graphs:

Constrained optimization

Starting structure

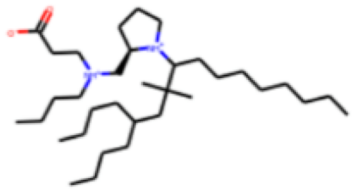


-8.32

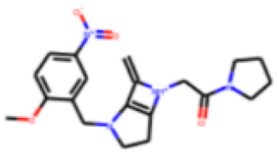


-5.55

Finished structure



-0.71



-1.78

(c) Constrained optimization of penalized logP

Summary of the talk

- Complex graphs can be successfully generated via sequential generation
- Each step a decision is made based on hidden state, which can be
  - Explicit: intermediate generated graphs, decode with GCN
  - Implicit: vector representation, decode with RNN
- Possible tasks:
  - Imitating a set of given graphs
  - Optimizing graphs towards given goals

Future Work

- Generating graphs in other domains
  - 3D shapes, social networks, etc.
- Simplify the optimization method:
  - Using MCMC instead of RL
- Scale up to large graphs:
  - Hierarchical action space, allowing high-level action like adding a structure at a time

PhD Students

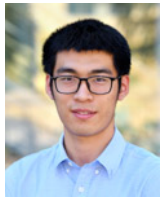

Alexandra Porter


Camilo Ruiz


Claire Donnat

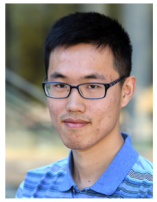

Emma Pierson


Weihua Hu


Jiaxuan You


Bowen Liu

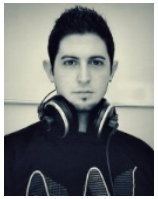

Mohit Tiwari


Rex Ying

Post-Doctoral Fellows


Baharan Mirzasoleiman

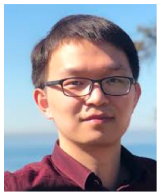

Marinka Zitnik


Michele Catasta

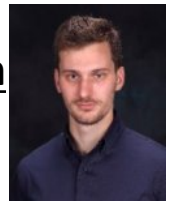

Robert Palovics


Shantao Li


Srijan Kumar


Hingwei Wang

Research Staff


Adrijan Bradaschia


Rok Sosic

Industry Partnerships

Funding

Collaborators

Dan Jurafsky, Linguistics, Stanford University
David Grusky, Sociology, Stanford University
Stephen Boyd, Electrical Engineering, Stanford University
David Gleich, Computer Science, Purdue University
VS Subrahmanian, Computer Science, University of Maryland
Sarah Kunz, Medicine, Harvard University
Russ Altman, Medicine, Stanford University
Jochen Profit, Medicine, Stanford University
Eric Horvitz, Microsoft Research
Jon Kleinberg, Computer Science, Cornell University
Sendhill Mullainathan, Economics, Harvard University
Scott Delp, Bioengineering, Stanford University
James Zou, Medicine, Stanford University

References

- Tutorial on Representation Learning on Networks at WWW 2018
- Inductive Representation Learning on Large Graphs. W. Hamilton, et al., NeurIPS 2017.
- Representation Learning on Graphs: Methods and Applications. W. Hamilton, et al. IEEE Data Engineering Bulletin, 2017.
- GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models. J. You, et al., *ICML*, 2018.
- Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation. J. You, et al., NeurIPS 2018.
- How Powerful are Graph Neural Networks? K. Xu, W. Hu, et al., ICLR 2019.
- Data & Code:
  - http://snap.stanford.edu/graphsage
  - https://github.com/bowenliu16/rl_graph_generation
  - https://github.com/williamleif/graphqembed
  - https://github.com/snap-stanford/GraphRNN