Efficient Graph Generation with Graph Recurrent Attention Networks

In **NeurIPS 2019**

Seonguk Seo

2018-20721

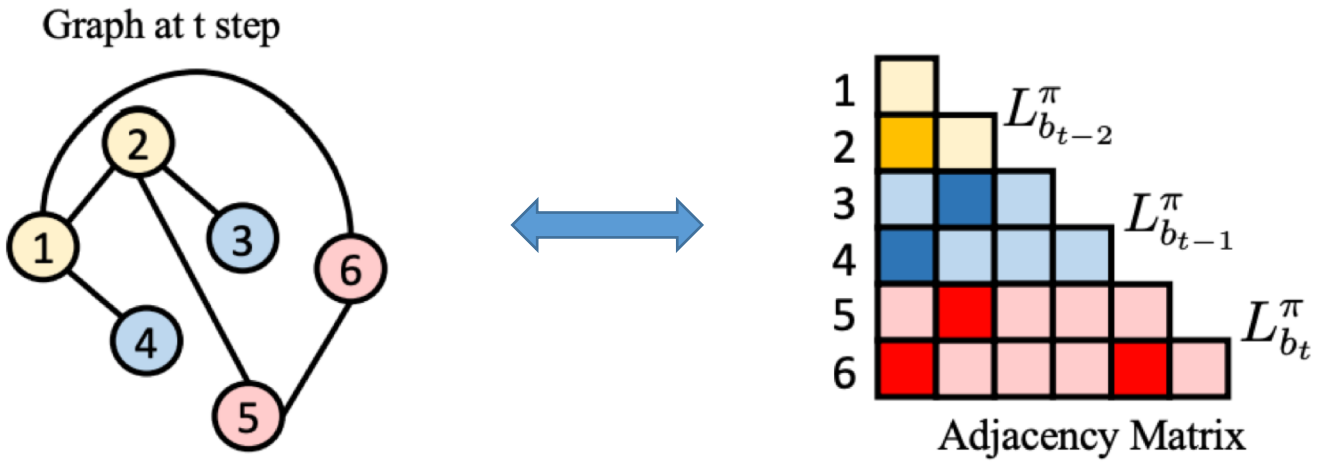
Seoul National University

Generative Model of Graphs

- Model the distribution of undirected graph $G = (V, E)$:

$$P(G) = \sum_{\pi} P(G, \pi) = \sum_{\pi} P(A_{\pi}) = \sum_{\pi} P(L_{\pi})$$

where π denotes a node ordering, A_{π} is an adjacency matrix and L_{π} denotes a lower triangular part of A_{π}.



Existing Work

- **Graph RNN** [You18]
  - It models graph generation as a sequential process, which accommodate complex dependencies between generated edges.
  - $O(N^2)$ for the best model (not scalable).
  - It has significant bottlenecks in handling long-term dependencies, and the results depend on node orderings.

[You18] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. **Graphrnn: Generating realistic graphs with deep auto-regressive models.** In ICML 2018

Contributions

- Their model consists of **O(N)** auto-regressive generation steps.
- Compared to RNN-based model, they propose an attention-based GNN that better utilizes the topology of the already generated graphs.
- They approximate the likelihood by marginalizing over a family of canonical node orderings.

Graph Recurrent Attention Networks (GRNN)

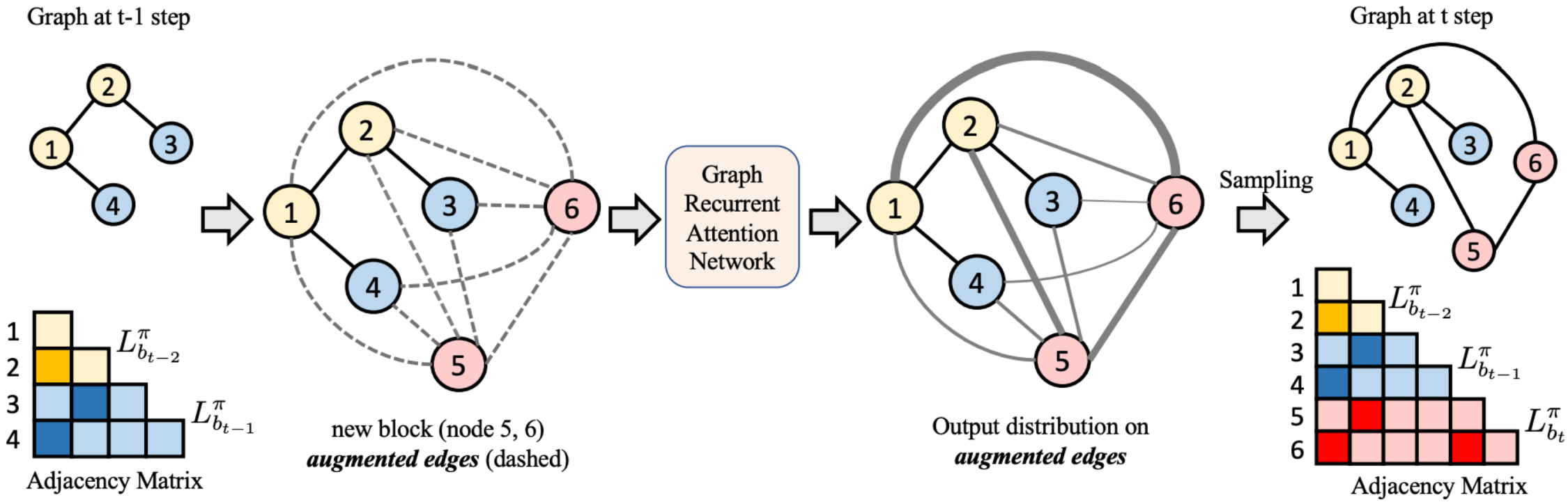
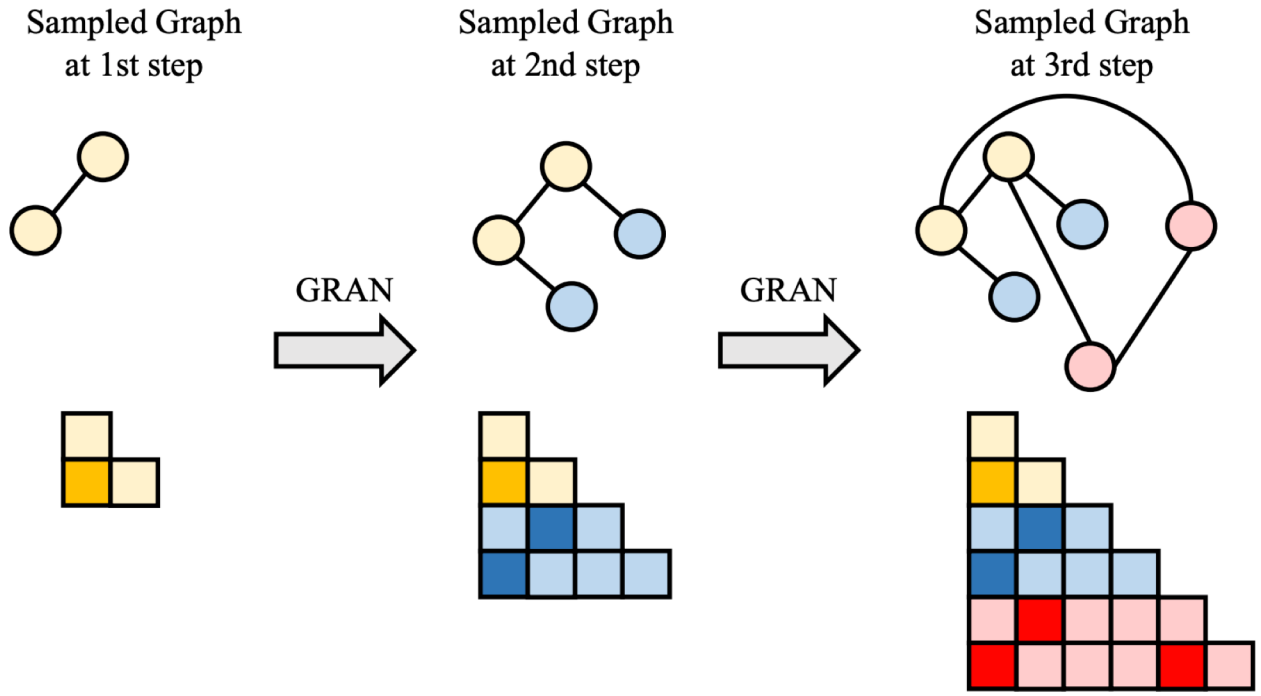


Figure 1: Overview of our model. Dashed lines are *augmented edges*. Nodes with the same color belong to the same block (block size = 2). In the middle right, for simplicity, we visualize the output distribution as a single Bernoulli where the line width indicates the probability of generating the edge.

Generation Process

- They generate one block of B rows of L_π for each time step.



- Breaking the dependency between generation steps allows parallel training and generated subgraph.
- Varying the block size permits the efficiency-quality trade-off.

Graph Recurrent Attention Networks (GRNN)

- Initial Node Representation

$$h^0_{\boldsymbol{b_i}} = W L^{\pi}_{\boldsymbol{b_i}} + b, \qquad \forall i < t.$$

- Message Passing

$$m^r_{ij} = f(h^r_i - h^r_j), \qquad a^r_{ij} = \text{Sigmoid}\left(g(\tilde{h}^r_i - \tilde{h}^r_j)\right),$$
$$\tilde{h}^r_i = [h^r_i, x_i], \qquad h^{r+1}_i = \text{GRU}(h^r_i, \sum\nolimits_{j \in \mathcal{N}(i)} a^r_{ij} m^r_{ij}).$$

- Output Distribution

$$p(L^{\pi}_{\boldsymbol{b_t}} | L^{\pi}_{\boldsymbol{b_1}}, ..., L^{\pi}_{\boldsymbol{b_{t-1}}}) = \sum_{k=1}^{K} \alpha_k \prod_{i \in \boldsymbol{b_t}} \prod_{1 \le j \le i} \theta_{k,i,j},$$
$$\alpha_1, \ldots, \alpha_K = \text{Softmax}\left(\sum\nolimits_{i \in \boldsymbol{b_t}, 1 \le j \le i} \text{MLP}_{\alpha}(h^R_i - h^R_j)\right),$$
$$\theta_{1,i,j}, \ldots, \theta_{K,i,j} = \text{Sigmoid}\left(\text{MLP}_{\theta}(h^R_i - h^R_j)\right)$$

Node Representation

- They first compute the initial node representations of the already-generated graph via a linear mapping

$$h^0_{\boldsymbol{b_i}} = W L^{\pi}_{\boldsymbol{b_i}} + b, \qquad \forall i < t.$$

- For current block, $h^0_{\boldsymbol{b_t}} = \mathbf{0}$
- In practice, computing $h^0_{\boldsymbol{b_{t-1}}}$ alone at t[th] generation step is enough, because $\{h^0_{\boldsymbol{b_i}} | i < t-1\}$ can be cached from previous steps, which reduces computation.

Graph Neural Network with Attentive Message Passing

- From the initial Node Representation, all edges associated with the current block are generated using a GNN.
- The r^{th} round of message passing is

$$m_{ij}^r = f(h_i^r - h_j^r), \qquad a_{ij}^r = \text{Sigmoid}\left(g(\tilde{h}_i^r - \tilde{h}_j^r)\right),$$

$$\tilde{h}_i^r = [h_i^r, x_i], \qquad h_i^{r+1} = \text{GRU}(h_i^r, \sum\nolimits_{j \in \mathcal{N}(i)} a_{ij}^r m_{ij}^r).$$

Output Distribution

- After R round message passing, they obtain the final node representation vector, and then model the probability of generating edges in the current block via a mixture of Bernoulli distributions.

$$p(L^{\pi}_{\boldsymbol{b_t}} | L^{\pi}_{\boldsymbol{b_1}}, ..., L^{\pi}_{\boldsymbol{b_{t-1}}}) = \sum_{k=1}^{K} \alpha_k \prod_{i \in \boldsymbol{b_t}} \prod_{1 \le j \le i} \theta_{k,i,j},$$

$$\alpha_1, \ldots, \alpha_K = \text{Softmax}\left(\sum\nolimits_{i \in \boldsymbol{b_t}, 1 \le j \le i} \text{MLP}_{\alpha}(h_i^R - h_j^R) \right),$$

$$\theta_{1,i,j}, \ldots, \theta_{K,i,j} = \text{Sigmoid}\left(\text{MLP}_{\theta}(h_i^R - h_j^R) \right)$$

Approximated Likelihood

- They aim to maximize a lower bound:

$$\log p(G) \geq \log \sum_{\pi \in \mathcal{Q}} p(G, \pi)$$

  where the size Q achieve a tradeoff between tightness of the bound (usually correlated with better model quality) and computational cost.

- They adopts DFS, BFS, k-core and degree descent ordering.

Experiments

- Dataset
  - Grid : 100 standard 2D grid graphs with $100 < |V| < 400$
  - Protein : 918 protein graphs with $100 < |V| < 500$
  - Point Cloud : 41 simulated 3D point clouds with $|V|_{avg} >= 1k$
- 64% train split, 16% valid split, 20% test split

Evaluation Metrics

- Compare the distributions of graph statistics between the generated and ground-truth graphs, by computing the maximum mean discrepancy (**MMD**) over the following 4 statistics.

1. Degree distributions
2. Clustering coefficient distributions
3. The number of occurrences of all orbits with 4 nodes
4. Spectra of the graphs (the eigenvalues of normalized graph Laplacian)

Visualization

- Sample graphs generated by GRAN with its comparison.

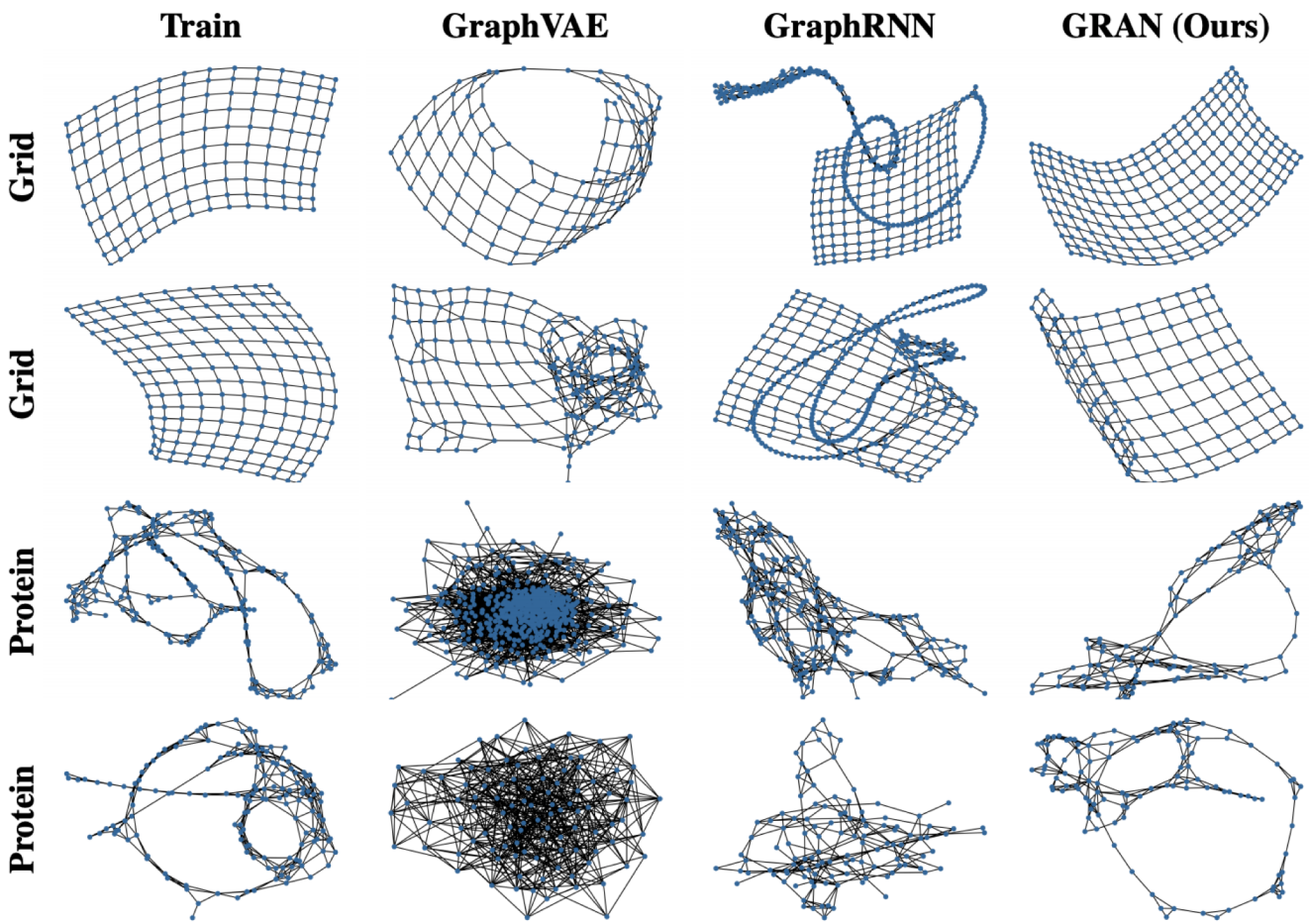


Figure 2: Visualization of sample graphs generated by different models.

Results

- For all metrics, lower value is preferred.

	Grid				Protein				3D Point Cloud			
	$\|V\|_{\max} = 361$, $\|E\|_{\max} = 684$ $\|V\|_{\text{avg}} \approx 210$, $\|E\|_{\text{avg}} \approx 392$				$\|V\|_{\max} = 500$, $\|E\|_{\max} = 1575$ $\|V\|_{\text{avg}} \approx 258$, $\|E\|_{\text{avg}} \approx 646$				$\|V\|_{\max} = 5037$, $\|E\|_{\max} = 10886$ $\|V\|_{\text{avg}} \approx 1377$, $\|E\|_{\text{avg}} \approx 3074$			
	Deg.	Clus.	Orbit	Spec.	Deg.	Clus.	Orbit	Spec.	Deg.	Clus.	Orbit	Spec.
Erdos-Renyi	0.79	2.00	1.08	0.68	$5.64e^{-2}$	1.00	1.54	$9.13e^{-2}$	0.31	1.22	1.27	$4.26e^{-2}$
GraphVAE*	$7.07e^{-2}$	$7.33e^{-2}$	0.12	$1.44e^{-2}$	0.48	$7.14e^{-2}$	0.74	0.11	-	-	-	-
GraphRNN-S	0.13	$3.73e^{-2}$	0.18	0.19	$4.02e^{-2}$	$\mathbf{4.79e^{-2}}$	0.23	0.21	-	-	-	-
GraphRNN	$1.12e^{-2}$	$\mathbf{7.73e^{-5}}$	$\mathbf{1.03e^{-3}}$	$\mathbf{1.18e^{-2}}$	$1.06e^{-2}$	0.14	0.88	$1.88e^{-2}$	-	-	-	-
GRAN	$\mathbf{8.23e^{-4}}$	$3.79e^{-3}$	$1.59e^{-3}$	$1.62e^{-2}$	$\mathbf{1.98e^{-3}}$	$4.86e^{-2}$	**0.13**	$\mathbf{5.13e^{-3}}$	$\mathbf{1.75e^{-2}}$	**0.51**	**0.21**	$\mathbf{7.45e^{-3}}$

Table 1: Comparison with other graph generative models. For all MMD metrics, the smaller the better. *: our own implementation, -: not applicable due to memory issue, Deg.: degree distribution, Clus.: clustering coefficients, Orbit: the number of 4-node orbits, Spec.: spectrum of graph Laplacian.

Efficiency vs Sample Quality

- The main limiting factor for graph generation speed is the number of generation steps T, which is related to the block size B.
- If B grows, which can improve speed, the model quality may suffer.
- They propose “stride sampling”, where neighboring blocks have an overlap