Deep Graph Translation

Xiaojie Guo
xguo7@gmu.edu
George Mason University
Fairfax, VA

Lingfei Wu
wuli@us.ibm.com
IBM Research
Yorktown Heights, NY

Liang Zhao
lzhao9@gmu.edu
George Mason University
Fairfax, VA

ABSTRACT

Deep graph generation models have achieved great successes recently, among which however, are typically unconditioned generative models that have no control over the target graphs given an input graph. In this paper, we propose a novel Graph-Translation-Generative-Adversarial-Networks (GT-GAN) that transforms the input graphs into their target output graphs. GT-GAN consists of a graph translator equipped with innovative graph convolution and deconvolution layers to learn the translation mapping considering both global and local features, and a new conditional graph discriminator to classify target graphs by conditioning on input graphs. Extensive experiments on multiple synthetic and real-world datasets demonstrate that our proposed GT-GAN significantly outperforms other baseline methods in terms of both effectiveness and scalability. For instance, GT-GAN achieves at least 10X and 15X faster runtimes than GraphRNN and RandomVAE, respectively, when the size of the graph is around 50.

KEYWORDS

Graph generation, deep neural networks, graph translation.

1 INTRODUCTION

In recent years, deep learning on graphs has seen a surge of interests, especially for graph representa16 tion and recognition tasks such as graph classification [5, 7, 12, 15] and embedding [8, 9]. Most recently researchers started to explore using deep generative models for graph synthesis on practical applications such as designing of new chemical molecular structure [19] and social interaction modeling [23]. This has led to many of the recent advances in deep graph generative models where some of these approaches are domain dependent models [6, 14] while others are generic [10, 16, 18, 19], although most of these models can only work on small graphs with 40 or fewer nodes.

However, there are two main drawbacks of existing deep graph generative models. First, one significant limitation of the previous approaches is that most of these models are only suitable for small graphs with 40 or fewer nodes, which is mainly due to their one-node-per-step generation manner. More importantly, most of the existing graph generation models are unconditioned and thus ignore rich input graph information for generating a new graph. In many applications, it is crucial to guide the graph generation process by conditioning on an input graph, which can be cast as a graph translation learning problem–translating the input graph to the output graph.

One straightforward way is to build a translation system by using a graph encoder-decoder architecture. However, there are several challenges for this type of approaches: 1) how to learn one-to-more mapping between the input graph and the target graphs. Different from the plain graph generation problem, a conditional graph synthesis task is to learn a distribution of target graphs conditioning on the input graph, which aims to capture the underlying implicit properties of the graphs, such as their scale-free characteristic. 2) how to jointly learn both local and global information for translation. One needs to not only learn the translation mapping in the local information (i.e. neighborhood of each node), but also in the global property of the whole graph (e.g., node degree distribution 40 or graph density).

To address the aforementioned challenges, we present a novel neural network: Graph-Translation-Generative-Adversarial-Nets (GT-GAN). We first propose a conditional GAN consisting of an encoder-decoder translator and a conditional graph discriminator to learn the one-to-more mapping (a conditional distribution) for graph translation. To jointly embed the local and global information, we present a novel graph encoder including both the edge and the node convolution layers. In addition, we further propose a novel graph U-net with graph skips and dedicated graph deconvolution layers including both the edge and the node deconvolution layers. Finally, GT-GAN is scalable with at most quadratic computation and memory consumption in terms of the number of nodes in a graph, making it suitable for at least modest-scale graphs (with hundreds of nodes, compared to the tens of nodes in most of existing graph generative models). Our code and data are available at https://github.com/anonymous1025/Deep-Graph-Translation-.

2 RELATED WORKS

Most of the existing GNN based graph generation for general graphs have been proposed in the last two years and are based on VAE [18, 19] and generative adversarial nets (GANs) [3], among others [16, 23]. Most of these approaches generate nodes and edges sequentially to form a whole graph, leading to the issues of being sensitive to the generation order and very time-consuming for large graphs. Differently, GraphRNN [23] builds an autoregressive generative model on these sequences with LSTM model and has demonstrated its good scalability. A variety of graph-to-sequence algorithms have been proposed. Applications of graph-to-sequence algorithms include text generation [2, 20, 22], graph algorithms and bAbI tasks [15, 20], among other. Recently, Sun et al. proposed a graphRNN based model which generates a graph topology based

KDD '19 workshop, August 04–08, 2019, Anchorage, Alaska

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

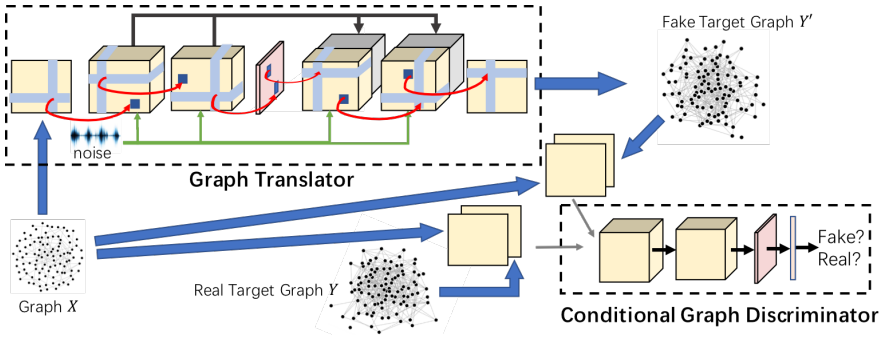

Figure 1: Architecture of GT-GAN consisting of graph translator and conditional graph discriminator. New graph encoder and decoder are specially designed for graph translation.

on another graph [21], which is contemporary with our work. However, it has no control on the size of the generated graphs and is difficult to scale to even modest-scale graph due to its one-node-per-step generation manner.

3 GT-GAN

Our goal is to learn an end-to-end translation mapping from an *input graph* to a *target graph*. Let an input graph $G_X = (\mathcal{V}, \mathcal{E}, A, S)$ such that $\mathcal{V}$ is the set of N nodes, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges, and $A \in \mathbb{R}^{N \times N}$ is an adjacency matrix (binary or weighted), where G_X can be weighted or unweighted, directed or undirected. Let $S \in \mathbb{R}^{N \times F}$ be a matrix with each line representing a node feature vector S_i. Denote $e_{i,j} \in \mathcal{E}$ as an edge from the node $v_i \in \mathcal{V}$ to $v_j \in \mathcal{V}$; $A_{i,j} \in A$ therefore denotes the corresponding weight of the edge $e_{i,j}$. Similarly, we define a *target graph* $G_Y = (\mathcal{V}', \mathcal{E}', A', S')$ that shares the same node sets and node features with G_X but with different topology and connection weights.

Formally, *graph translation* is to learn a translator from an input graph $G_X \in \mathcal{G}_X$ with a random noise U to generate a target graph $G_Y \in \mathcal{G}_Y$, where $\mathcal{G}_X$ and $\mathcal{G}_Y$ denote the domains of the input and target graphs, respectively. The translation mapping is denoted as $\mathcal{T} : U, G_X \rightarrow G_Y$.

Overall Architecture and Loss function. We propose the Graph-Translation GAN (GT-GAN) that consists of a graph translator $\mathcal{T}$ and a conditional graph discriminator $\mathcal{D}$, as shown in Fig.1. $\mathcal{T}$ is trained to produce target graphs that cannot be distinguished from "real" graphs by $\mathcal{D}$, that is, distinguishing the generated target graph $G_{Y'} = \mathcal{T}(G_X, U)$ from the real one G_Y based on the current input graph G_X. $\mathcal{T}$ and $\mathcal{D}$ undergo an adversarial training process based on input and target graphs shown as below:

$$\begin{aligned}\mathcal{L}(\mathcal{T}, \mathcal{D}) =& \mathbb{E}_{G_X, G_Y}[\log \mathcal{D}(G_Y | G_X)] \\ &+ \mathbb{E}_{G_X, U}[\log(1 - \mathcal{D}(\mathcal{T}(G_X, U) | G_X))]\end{aligned} \tag{1}$$

where $\mathcal{T}$ tries to minimize this objective against an adversarial $\mathcal{D}$ that tries to maximize it, i.e. $\mathcal{T}^* = \arg\min_{\mathcal{T}} \max_{\mathcal{D}} \mathcal{L}(\mathcal{T}, \mathcal{D})$. An L1 loss is applied to the weight adjacent matrix of generated target graphs and real target graphs:

$$\mathcal{L}_{l1}(\mathcal{T}) = \mathbb{E}_{W, W', U}[\|W' - \mathcal{T}(W, U)\|_1] \tag{2}$$

The training process is a trade-off game between $\mathcal{L}_{l1}$ and $\mathcal{L}(\mathcal{T}, \mathcal{D})$, which jointly enforces $\mathcal{T}(G_X, U)$ and G_Y to follow a similar topological pattern but may not necessarily the same. Thus, the final objective is:

$$\mathcal{T}^* = \arg\min_{\mathcal{T}} \max_{\mathcal{D}} \mathcal{L}(\mathcal{T}, \mathcal{D}) + \mathcal{L}_{l1}(\mathcal{D}) \tag{3}$$

where $\mathcal{T}^*$ is the optimal graph translator which generates graphs that are as "real" as possible.

The graph translator $\mathcal{T}$ is an encoder-decoder architecture, where we propose a new graph encoder to obtain the node representations of the input graph and propose the graph deconvolution with skips to generate the target graph, as shown in Fig.1, which we elaborated in the followings sections.

Edge Convolutions Layer of Graph Encoder. To learn the local information, the proposed encoder learns each node representation based on its n-hop neighbors. To learn the global information, it learns each node representation by looking for more "virtual neighbors" regarding the latent relations from the whole graph. Thus, we propose the "edge convolution" layer to learn a group of multi-mode relations from the topology of the input graph, which can include both the n-hop connections and the latent relations that are derived from their adjacent edges/relations as shown in Fig.2(a). And then the "node convolution" layer is used to embed each node representations by aggregating its "virtual neighbors" that related to each latent relations, as shown in Fig.2(b).

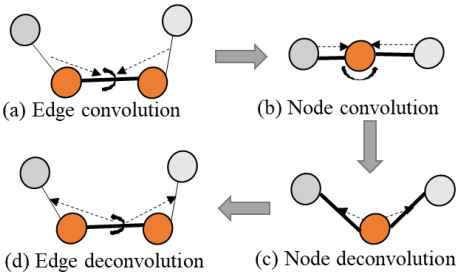

Figure 2: Graph convolution and deconvolution

In each "edge convolution" layer, each node pair's latent relation is computed by its adjacent edges or the extracted adjacent relations from the last layer. In the directed graph, each node have in-coming edge(s) and out-going edge(s). Thus, there are two learnable parametric vectors ϕ and ψ as convolution filters for two directions to convolute the adjacent edges/relations for each node pairs. The relation $E_{i,j}^{l,m}$ in the mth relation mode of the lth layer is learned by the out-going edges/relations of node v_i and the in-coming edges/relations of node v_j,

$$E_{i,j}^{l,m} = \sum_{n=1}^{R_{l-1}} (\sigma(\sum_{k_1=1}^{N} E_{i,k_1}^{l-1,n} \phi_{k_1}^{l,m}) + \sigma(\sum_{k_2=1}^{N} E_{k_2,j}^{l-1,n} \psi_{k_2}^{l,m}))$$

where $E_{i,j}^{1,1} \equiv A$ and $\phi^{l,m} \in \mathbb{R}^{N\times 1}$ refers to the filter vector to be learned and $\phi_{k_1}^{l,m}$ refers to the element of $\phi^{l,m}$ that is related to node v_{k_1}. R_{l-1} refers to the number of relation modes extracted for the $(l-1)$th layer of the graph encoder. σ refers to the activation function RELU for all hidden layers.

Node Convolutions Layer of Graph Encoder . After learning the various modes of relations, the "node convolution" layer learns each node's representations by aggregating its "virtual neighbors" in terms of each mode of relation. The mth feature vector of node representation tensor $\bar{H}_i^m \in \mathbb{R}^{1\times F}$ for node v_i is computed as:

$$\bar{H}_i^m = \sum_{n=1}^{R_{l-1}} (\sigma(\sum_{k_1=1}^{N} E_{i,k_1}^{l-1,n} \mu_{k_1}^m S_{k_1}) + \sigma(\sum_{k_2=1}^{N} E_{k_2,i}^{l-1,n} \nu_{k_2}^m S_{k_2})),$$

where $\bar{H}_i \in \mathbb{R}^{R_l\times F}$ and R_l refers to the number of feature vectors in the "node comvolution" layer. Here $\mu^m, \nu^m \in \mathbb{R}^{N\times 1}$ refer to the filter vectors for the two directions to be learned and $\mu_{k_1}^m$ refers to the element of μ^m that is related to node v_{k_1}. $\bar{H}_i$ is then flattened and transformed into a node representation vector $H_i \in \mathbb{R}^{1\times C}$ by a fully connected layer. C is the length of the node representation. Note that our graph encoder is designed for a directed graph, and it is easily generalized to an undirected graph, where the weight vector is shared by both directions.

Node Deconvolution Layer of Graph Decoder. The proposed graph deconvolution technique incorporates both "node deconvolution" and "edge deconvolution" layers.

First, the "node deconvolution" layer are used to generates the latent multi-mode relations of the target graph based on the learned latent node representations. As shown in Fig. 2(c), "node deconvolution" is a reversed process of the "node" convolution. Since each node has an influence to its relations connecting to other nodes. Then the relation $E_{i,j}^{1,m}$ between node v_i and node v_j in the mth relation mode of the lth "node" deconvolution layer in the decoder can be computed as follows:

$$E_{i,j}^{1,m} = \sum_{n=1}^{C} (\sigma(H_i^n \bar{\mu}_j^m) + \sigma(H_j^n \bar{\nu}_i^m)), \tag{4}$$

where $\sigma(H_i^n \bar{\mu}_j^m)$ means the deconvolution contribution of node v_i to its relation with node v_j made by the nth element of its node representations, and $\bar{\mu}_j^m$ represents the element of the deconvolution filter vector $\bar{\mu}^m \in \mathbb{R}^{1\times N}$ that is related to node v_j.

Edge Deconvolution Layer of Graph Decoder. We can now recursively apply our proposed "edge deconvolution" layer to decode the latent relation between each pair of nodes from the upper layer to those of lower layer. As a reversed way of "edge" convolution, the relation of each pair of nodes in the $(l-1)$th layer can make contribution to generating itself and its adjacent relations in the lth layer, as shown in Fig. 2(d). Thus, the relation $E_{i,j}^{l,m}$ between node v_i and node v_j in the lth layer is computed as follows:

$$E_{i,j}^{l,m} = \sum_{n=1}^{R'_{l-1}} (\sigma(\bar{\phi}_j^{l,m} \sum_{k_1=1}^{N} E_{i,k_1}^{l-1,n}) + \sigma(\bar{\psi}_j^{l,m} \sum_{k_2=1}^{N} E_{k_2,j}^{l-1,n})),$$

where $\bar{\phi}^{l,m} \sum_{k_1=1}^{N} E_{i,k_1}^{l-1,n}$ is interpreted as the decoded contribution of node i to its relations with node v_j, and $\bar{\phi}^{l,m}$ refers to the element of deconvolution filter vector that is related to node v_j. R'_{l-1} refers to the number of relation modes extracted by the $(l-1)$th layer in the graph decoder. The output of the last "edge" deconvolution layer denotes the edges of the target graph.

Skips for graph deconvolution. Based on the graph deconvolution above, it is possible to utilize skips to link the extracted latent relation sets of each layers in the graph encoder with those in the graph decoder. Specifically, the output of the lth "edge deconvolution" layer with R_l channels in the decoder is concatenated with the output of the lth "edge convolution" layer with R'_l channels in encoder to form joint $R_l + R'_l$ channels, which are then input into the $(l+1)th$ deconvolution layer.

Conditional Graph Discriminator The graph discriminator must distinguish between the "translated" target graph and the "real" ones based on the input graphs, as this helps to train the generator in an adversarial way. Technically, this requires the discriminator to accept two graphs simultaneously as inputs (a target graph and an input graph or a generated graph and an input graph), and classify the two graphs as either related or not. Thus, we propose a *conditional graph discriminator* (CGD) which leverages the same graph convolution layers in the translator for the graph classification, as shown in Fig.1. Specifically, the input and target graphs are both ingested by CGD and stacked into a $N \times N \times 2$ tensor which can be considered a 2-channel input. After obtaining the node representations, the graph-level embedding is computed by summing these node embeddings. Finally, a softmax layer is implemented to distinguish the input graph-pair from the real graph or generated graph.

4 EXPERIMENT

4.1 Experimental Settings

4.1.1 Datasets. Two groups of synthetic datasets are used: Scale-free graphs set and Poisson-random graphs set. Each group has several datasets with different graph sizes. Each dataset consists of 5000 pairs of input and target graphs: half for training and the remaining half for validating. For **Scale-free Graph**, each input graph is generated as a directed scale-free network, whose degree distribution follows power-law property [4]. To construct target graphs, each weight between two connected nodes in input graph will be added by $m > 1$. For **Poisson-random graphs**, each input graph is a directed growing random network generated by [13]. Then for an input graph with $|E|$ number of edges, $k|E|$ number of edges are randomly added to form the target graph, where k follows the Poisson distribution with the mean of 5. We also test our model to forecast future potential malicious authentication graphs given the user's normal authentication graph. Each **user authentication graph** is a directed weighted graph, where nodes represent computers and the weights of the edges represent the authentication activities at certain frequencies. There are 78 pairs of graphs (malicious and normal behavior) of graph size 50 and 315 pairs of graphs of graph size 300 from 97 users in two subsets. We performed a 2-fold cross-validations and 3-fold cross-validation, respectively, for the two subsets. We also test our model on the **IOT network malware** confinement prediction (predicting optimal network operation given a compromised one). There are three subsets of graph pairs with different sizes (20, 40 and 60), where the nodes represent devices and the node attributes indicating the compromised status of the nodes. The weights of the edges represent the distance between two devices. There are 334 pairs of input (compromised IOT) and target graphs (optimal IOT) in

each subset and each is divided into two parts for the 2-fold cross validation.

4.1.2 Comparison Methods. Since there is no existing work on deep graph translation, we compare against current state-of-the-arts of graph generation methods: GraphRNN [23], GraphVAE [19], GraphGMG [16], RandomVAE [18], and S-Generator which is a supervised deterministic graph generation using only our graph translator with L1 loss. All the comparison methods are directly trained by the real target graphs without the conditions of input graphs as they can only do graph generation instead of translation. The learning rates for GraphGMG, GraphVAE, and RandomVAE are 0.001 and the learning rate of GraphRNN is 0.003.

4.2 Performance

4.2.1 Model scalability Analysis. Fig.3 illustrates the scalability of GT-GAN against two methods (GraphRNN and RandomVAE) in terms of memory consumption and computational time, respectively. As shown in Fig. 3, when the graph size increases up to 50, the memory consumption of the GT-GAN maintains almost constant and computational time grows slowly, both of which are less than 1500MB and 300 seconds, respectively. In contrast, the memory consumption and computational time of RandomVAE increases super-linearly as the graph size increases, making it hard to scale even to the graph size of 50. Though the, memory consumption and runtime of GraphRNN increases slightly as the graph size increases, their costs were almost two times larger in memory requirement and ten times slower in runtime than the proposed GT-GAN.

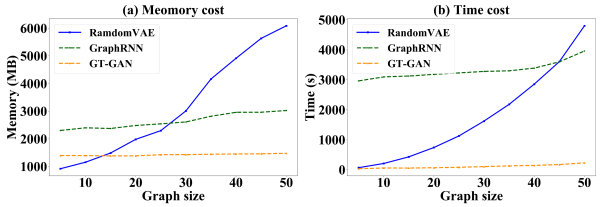

Figure 3: Scalability plots on memory and time cost of GT-GAN, RandomVAE and GraphRNN

4.2.2 Results for the synthetic datasets. We diretly evaluated the sparsity similarity between the generated and real target graphs in terms of the node degree distribution. The four metrics used to measure the correlation between two distributions were the Jensen-Shannon distances (JS), the Hellinger Distance (HD), the Bhattacharyya Distance (BD) and the Wasserstein Distances (WD).

Table 1 shows the distances between generated and real target graphs in terms of node degree on the above metrics. The entry "Inf" represents the distance more than 1000. The proposed GT-GAN outperforms all baselines in almost every metrics, especially in Bhattacharyya distance (by average 15%) and Hellinger distance for all graph sizes. Fig. 4 shows the node degree distribution curve of three generated and real target graphs by GT-GAN. The curves of the generated graphs follow power-law rule correctly and become more and more close to the real target graphs as the graph size increases. Similar observations in direct evaluation results (e.g. average degree, repository and density) of Poisson random graphs and user authentication graphs can be found in Appendix B and C.

Table 1: Node degree distribution distance between the generated and real graphs for scale-free graphs

Graph size	Methods	JS	HD	BD	WD
10	Random-VAE	0.42	**0.98**	Inf	7.58
	GraphRNN	0.47	**0.98**	Inf	1.64
	GraphVAE	0.67	1.00	Inf	2.85
	GraphGMG	0.43	**0.98**	Inf	1.69
	S-Generator	**0.35**	0.98	3.45	0.80
	GT-GAN	**0.35**	**0.98**	**3.44**	**0.77**
20	RandomVAE	0.51	0.97	Inf	1.74
	GraphRNN	0.50	0.98	Inf	1.44
	S-Generator	0.36	**0.96**	2.84	0.67
	GT-GAN	**0.35**	**0.96**	**2.74**	**0.66**
100	GraphRNN	0.48	0.88	Inf	0.90
	S-Generator	**0.14**	0.68	0.64	**0.30**
	GT-GAN	0.15	**0.43**	**0.24**	0.31
150	GraphRNN	0.42	0.74	Inf	0.95
	S-Generator	0.08	0.31	**0.11**	0.29
	GT-GAN	**0.07**	**0.30**	**0.11**	**0.27**

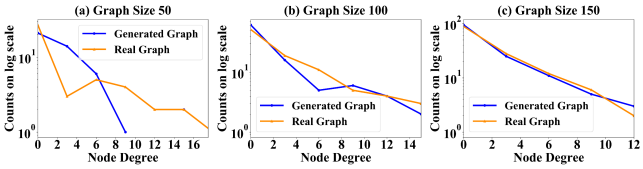

Figure 4: Examples of node degree distributions of generated and target graphs for scale-free graphs

4.2.3 Results for the user authentication datasets. The performance on the user authentication datasets is evaluated by the indirect evaluation. In indirect evaluation, the validation sets are split evenly into two subsets. The first subset is for training a graph classifier proposed in [17], using only the negative samples plus the generated target graphs. In addition, a "gold standard" classifier is trained based on negative samples and *real* target graphs. The second subset containing both the input and real target graphs validate the classifiers trained above. Table 2 shows the average results of graph classifiers for different methods on the user authentication graphs. Classifiers trained by the graphs generated by GT-GAN can effectively classify normal and hacked behaviors with average AUC above 0.78 and performs consistently better than others when graph size varies from 50 to 300. We refer readers to the detailed evaluation process and more indirect results in Appendix D.

Table 2: Results for user authentication graphs

Graph size	Method	Precision	Recall	AUC	F1
50	RandomVAE	0.32	0.51	0.26	0.39
	GraphRNN	0.34	0.36	0.50	0.36
	S-Generator	0.72	0.61	0.74	0.66
	GT-GAN	**0.79**	**0.68**	**0.78**	**0.73**
	Gold Standard	0.97	0.97	0.97	0.97
300	S-Generator	0.77	0.58	0.62	0.66
	GT-GAN	**0.84**	**0.66**	**0.79**	**0.74**
	Gold Standard	0.98	0.96	0.97	0.97

4.2.4 Results on IOT dataset. Table 3 compared the performance of GT-GAN and comparison methods for the IOT dataset by examining the edges of the generated and real target graphs for four metrics:

MSE (mean squared error), R2 (coefficient of determination score), Pearson Correlation (P) of adjacent matrix, and ACC (Accuray) for the correct existence of edges among all the pairs of nodes. The results show that GT-GAN performed almost the best for all the three subsets. GT-GAN got highest Pearson Correlation of around 0.8 for all three subsets compared to the other methods which had Pearson Correlations below 0.4.

Table 3: Results for the IOT datasets

Size	Method	R2	MSE	P	ACC (%)
20	GraphRNN	0.16	1775.58	0.23	83.97
	GraphVAE	0.39	2109.64	0.32	81.19
	GT-GAN	**0.67**	**370.91**	**0.85**	**92.00**
40	GraphRNN	0.44	1950.46	0.29	70.54
	GraphVAE	**0.73**	2410.57	0.16	66.60
	GT-GAN	0.69	**408.50**	**0.86**	**93.94**
60	GraphRNN	0.52	1831.43	0.04	61.07
	GraphVAE	0.00	2453.61	0.04	50.64
	GT-GAN	**0.62**	**566.88**	**0.80**	**94.63**

4.2.5 Ablation Experiments on the Encoders and Decoder. To further validate the superiority of the proposed graph convolution and deconvolution layer, an ablation experiment was conducted by replacing the encoder and decoder with node embedding and decoder methods normally used. The graph encoder was replaced by the GCN [12] and DCNN [1], both of which consider edge and node features, while the graph decoder was replaced by the decoder in VGAE [11]. Table. 4 shows the results of the ablation study on part of the scale-free (Scale), user authentication (Auth) and IOT datasets. The encoder of GT-GAN outperformed both the GCN- and DCNN- based encoders by a large margin on these datasets. For example, on Auth-I, GT-GAN performed 43%, 50%, 31%, and 38% better on average, when compared with the GCN and DCNN encoders in terms of precision, recall, AUC and F1-scores, respectively. In addition, the decoder in GT-GAN was deemed both effective and irreplaceable for graph generation. For example, on IOT-III, GT-GAN performed 6.97%, 45.00%, and 83.33% better than the decoder in VGAE in terms of ACC, P and R2, respectively.

Table 4: Ablation study on four datasets

Dataset	Method	JS	HD	BD	WD
Scale-III	GCN+decoder	0.18	0.48	0.27	18.84
	DCNN+decoder	0.65	0.96	Inf	0.77
	Encoder+VGAE	0.31	0.63	0.51	43.78
	GT-GAN	**0.15**	**0.43**	**0.24**	**0.31**
		P	R	AUC	F1
Auth-I	GCN+decoder	0.31	0.35	0.52	0.33
	DCNN+decoder	0.59	0.55	0.55	0.57
	Encoder+VGAE	0.49	0.46	0.61	0.47
	GT-GAN	**0.79**	**0.68**	**0.78**	**0.73**
Auth-II	DCNN+decoder	0.58	0.42	0.62	0.51
	GT-GAN	**0.84**	**0.66**	**0.79**	**0.74**
		R2	MSE	P	ACC(%)
IOT-III	GCN+decoder	0.46	818.25	0.71	92.69
	DCNN+decoder	0.52	721.98	0.74	93.26
	Encoder+VGAE	0.12	1337.16	0.44	88.14
	GT-GAN	**0.62**	**566.88**	**0.80**	**94.63**

5 CONCLUSION AND FUTURE WORKS

This paper focuses on a new problem: deep graph translation. To achieve this, we propose a novel GT-GAN which translates an input graph to a target graph. To learn both global and local mapping between graphs, a new graph encoder-decoder model have been proposed while preserving the graph patterns in various scales. Experimental results show that our GT-GAN can discover the ground-truth translation rules, and significantly outperform other baselines. This paper opens a thread of research for deep graph translation in many practical applications.

REFERENCES

[1] James Atwood and Don Towsley. 2016. Diffusion-convolutional neural networks. In *NeurIPS*. 1993–2001.

[2] Daniel Beck, Gholamreza Haffari, and Trevor Cohn. 2018. Graph-to-sequence learning using gated graph neural networks. *arXiv preprint arXiv:1806.09835* (2018).

[3] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. 2018. NetGAN: Generating Graphs via Random Walks. *arXiv preprint arXiv:1803.00816* (2018).

[4] Béla Bollobás, Christian Borgs, Jennifer Chayes, and Oliver Riordan. 2003. Directed scale-free graphs. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 132–139.

[5] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann Lecun. 2014. Spectral networks and locally connected networks on graphs. In *ICLR*.

[6] Hanjun Dai, Yingtao Tian, Bo Dai, Steven Skiena, and Le Song. 2018. Syntax-Directed Variational Autoencoder for Structured Data. *arXiv preprint arXiv:1802.08786* (2018).

[7] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS*. 3844–3852.

[8] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD*. ACM, 855–864.

[9] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*. 1025–1035.

[10] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. 2018. Junction Tree Variational Autoencoder for Molecular Graph Generation. *arXiv preprint arXiv:1802.04364* (2018).

[11] Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308* (2016).

[12] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

[13] Paul L Krapivsky and Sidney Redner. 2001. Organization of growing random networks. *Physical Review E* 63, 6 (2001), 066123.

[14] Matt J Kusner, Brooks Paige, and José Miguel Hernández-Lobato. 2017. Grammar Variational Autoencoder. In *ICML*. 1945–1954.

[15] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2016. Gated graph sequence neural networks. *ICLR* (2016).

[16] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. 2018. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324* (2018).

[17] Giannis Nikolentzos, Polykarpos Meladianos, Antoine Jean-Pierre Tixier, Konstantinos Skianis, and Michalis Vazirgiannis. 2017. Kernel Graph Convolutional Neural Networks. *arXiv preprint arXiv:1710.10689* (2017).

[18] Bidisha Samanta, Abir De, Niloy Ganguly, and Manuel Gomez-Rodriguez. 2018. Designing Random Graph Models Using Variational Autoencoders With Applications to Chemical Design. *arXiv preprint arXiv:1802.05283* (2018).

[19] Martin Simonovsky and Nikos Komodakis. 2018. GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders. *arXiv preprint arXiv:1802.03480* (2018).

[20] Linfeng Song, Yue Zhang, Zhiguo Wang, and Daniel Gildea. 2018. A graph-to-sequence model for amr-to-text generation. *arXiv preprint arXiv:1805.02473* (2018).

[21] Mingming Sun and Ping Li. 2019. Graph to Graph: a Topology Aware Approach for Graph Structures Learning and Generation. In *AISTATS*. 2946–2955.

[22] Kun Xu, Lingfei Wu, Zhiguo Wang, and Vadim Sheinin. 2018. Graph2Seq: Graph to Sequence Learning with Attention-based Neural Networks. *arXiv preprint arXiv:1804.00823* (2018).

[23] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. 2018. GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models. In *ICML*. 5694–5703.

A MORE EXPERIMENTAL RESULTS FOR SCALE FREE GRAPH SET

Fig. 9 shows 18 examples for scale free dataset from size 50 to 150. Table 5 shows the indirect evaluation on scale free dataset. The indirect evaluation strategy is the same with the one used on user authentication dataset.

Table 5: Indirect evaluation for scale-free graphs

Size	Method	P	R	AUC	F1
10	RandomVAE	0.83	0.29	0.31	0.42
	GraphRNN	0.31	0.11	0.49	0.16
	GraphVAE	0.75	0.23	**0.65**	0.35
	GraphGMG	0.42	0.12	0.49	0.18
	S-Generator	0.46	**0.83**	0.43	0.59
	GT-GAN	**1.00**	0.50	0.52	**0.67**
	Gold Standard	0.81	0.74	0.82	0.77
50	RandomVAE	0.89	0.67	0.84	0.76
	GraphRNN	0.52	0.53	0.70	0.52
	S-Generator	0.50	**1.00**	0.37	0.67
	GT-GAN	**0.93**	0.82	**0.94**	**0.87**
	Gold Standard	0.94	0.90	0.97	0.91
100	GraphRNN	0.61	0.65	0.67	0.60
	S-Generator	0.50	**1.00**	0.50	0.67
	GT-GAN	**0.72**	0.69	**0.68**	**0.70**
	Gold Standard	0.99	0.61	0.81	0.75
150	GraphRNN	0.73	**0.92**	0.92	0.81
	S-Generator	**1.00**	0.50	0.50	0.67
	GT-GAN	0.94	0.79	**0.96**	**0.86**
	Gold Standard	0.99	0.93	0.96	0.95

B MORE EXPERIMENTAL RESULTS FOR POISSON RANDOM GRAPH SET

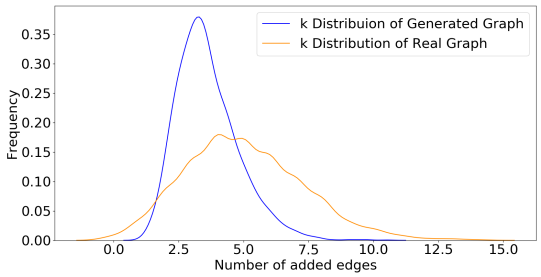

Figure 5: Distribution of k for generated graphs and real graphs in Poisson random graph set

For Poisson random graphs, the distributions of k in the real target graphs and those generated graphs are compared. The mean of edge increasing ratio k for generated graphs by our GT-GAN is 3.6, compared to the real value of 5, which implies that the GT-GAN generally is able to discover the underlying increasing ratio between input and target graphs. More evaluation results (e.g. degree and repository) can be found in Appendix B. We draw the probability density curve of the proportion k. Fig. 5 shows the distribution of the k in graphs generated by GT-GAN and the real graphs. The distribution plot is drew based on 3000 samples. Both of the two distribution have main degree values in the range from 2 to 7, while there is difference in the max frequency due to the limit of the samples amount. However, it prove that the proposed GT-GAN do learn the distribution type of translation parameter k in this task. Table 6 shows the indirect evaluation on Poisson random dataset. Table 7 shows the distance measurement between generated graphs and real graphs in several metrics. For the metric "degree", we use Wasserstein distances to measure the distance of two degree distribution. For other metrics, we calculate the MSE between generated graphs and real graphs.

Table 6: Indirect evaluation for poisson-random graphs

Size	Method	P	R	AUC	F1
10	RandomVAE	0.98	0.75	**0.99**	0.85
	GraphRNN	0.98	0.99	**0.99**	**0.98**
	GraphVAE	0.98	0.92	0.97	0.94
	GraphGMG	0.98	0.98	0.98	**0.98**
	S-Generator	0.50	**1.00**	0.50	0.66
	GT-GAN	**1.00**	0.87	0.94	0.90
	Gold Standard	0.99	1.00	1.00	0.99
50	RandomVAE	0.93	0.46	**1.00**	0.63
	GraphRNN	**1.00**	0.99	0.99	0.99
	S-Generator	0.49	0.98	0.35	0.65
	GT-GAN	**1.00**	**0.99**	**1.00**	**0.99**
	Gold Standard	0.99	1.00	1.00	0.99
100	GraphRNN	**1.00**	0.99	**1.00**	**0.99**
	S-Generator	0.50	**1.00**	0.51	0.66
	GT-GAN	0.90	**1.00**	**1.00**	0.94
	optimal	1.00	1.00	1.00	1.00
150	GraphRNN	0.95	0.99	**1.00**	0.96
	S-Generator	0.50	**1.00**	0.49	0.66
	GT-GAN	**0.97**	**1.00**	**1.00**	**0.98**
	Gold Standard	1.00	0.99	1.00	0.99

C MORE EXPERIMENTAL RESULTS FOR USER AUTHENTICATION GRAPH SET

About Original Dataset. This data set spans one calendar year of contiguous activity spanning 2012 and 2013. It originated from 33.9 billion raw event logs (1.4 terabytes compressed) collected across the LANL enterprise network of approximately 24,000 computers. Here we consider two sub dataset. First is the user log-on activity set. This data represents authentication events collected from individual Windows-based desktop computers, servers, and Active Directory servers. Another dataset presents specific events taken from the authentication data that present known red team compromise events, as we call malicious event. The red team data can used as ground truth of bad behavior which is different from normal user. Each graph can represent the log-on activity of one user in a time window. The event graphs are defined like this: The node refers to the computers that are available to a user and the edge represents the log-on activity from one computer to another computer of the user.

Table 7: MSE of Graph properties measurements for poisson random graphs

Graph size	Method	Density	Ave-Degree	Reciprocity
10	RandomVAE	0.1772	2.8172	0.3917
	GraphRNN	0.2665	2.2078	0.1344
	GrapgGMG	0.3519	2.4286	0.1338
	GraphVAE	0.2881	3.1986	0.3103
	S-Generator	**0.2993**	**1.5751**	**0.0737**
	GT-GAN	0.3084	1.7707	0.1327
50	RandomVAE	Inf	23.680	0.5362
	GraphRNN	**0.0110**	3.6000	0.0125
	S-Generator	0.0120	**2.9082**	0.0125
	GT-GAN	0.0155	3.2960	**0.0047**
100	GraphRNN	0.0123	3.5475	0.0034
	S-Generator	**0.0029**	**2.9167**	**0.0034**
	GT-GAN	0.0142	4.3730	0.0043
150	GraphRNN	**0.0012**	3.6619	0.0016
	S-Generator	0.0013	**2.9467**	**0.0016**
	GT-GAN	0.0061	5.0410	0.0019

Direct evaluation of User authentication Graph Set. For the user authentication graphs, the real target graphs and those generated are compared under well-recognized graph metrics including degree of nodes, reciprocity, and density. We calculate the distance of degree distribution and Mean Sqaured Error (MSE) for reciprocity and density. 8 shows the mean square error of the generated graphs and real graphs for all users.

Table 8: MSE of Graph properties measurements for user authentication dataset

Graph size	Method	Density	Reciprocity	Average Degree
50	RandomVAE	0.0005	0.0000	6.4064
	GraphRNN	0.0032	0.0000	2.7751
	S-Generator	0.0244	0.0342	24.130
	GT-GAN	**0.0003**	**0.0000**	**0.0002**
300	S-Generator	0.0113	0.0010	8.6839
	GT-GAN	**0.0004**	**0.0000**	**0.0006**

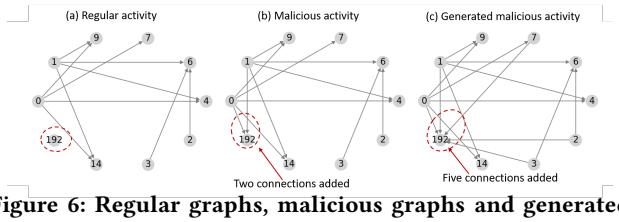

Figure 6: Regular graphs, malicious graphs and generated graphs of User 049

Case Studies on the generated target graphs. Fig. 6 shows the example of User 049 with regular activity graph, real malicious activity graph and malicious activity graph generated by our GT-GAN from left to right. Only those of edges with difference among them are drawn for legibility. It can be seen that, the hacker performed attacks on Computer 192, which has been successfully simulated by our GT-GAN. In addition, GT-GAN also correctly identified that the Computer 192 is the end node (i.e., with only incoming edges) in this attack. This is because GT-GAN can learn both the global hacking patterns (i.e., graph density, modularity) but also can learn local properties for specific nodes (i.e., computers). GT-GAN even successfully predicted that the hacker connect from Computers 0 and 1, with Computers 7 and 14 as false alarms.

For User006 in Fig. 7, the red team attackers make more connections on Node 36 compared to user's regular activity, as marked in red rectangle. GT-GAN leans how to choose the Node 36 and it generated more connections too in the Node 36

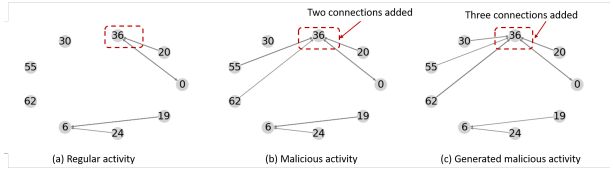

Figure 7: Regular graphs, malicious graphs and generated graphs for User 006

D FLOWCHART OF INDIRECT EVALUATION PROCESS

Fig. 8 shows the process of the indirect evaluation process.

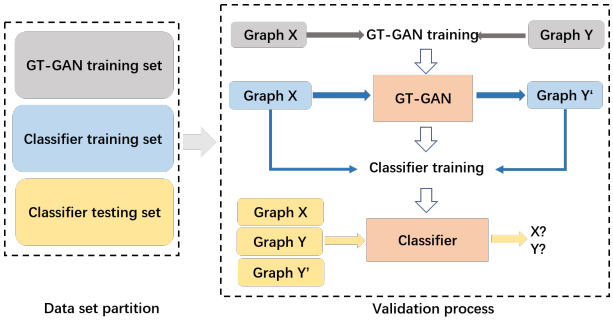

Figure 8: Flow chart of validation

E ARCHITECTURE PARAMETER FOR GT-GAN MODEL

Graph Generator: Given the graph size (number of nodes) N of a graph. The output feature map size of each layer through graph generator can be expressed as:

$N \times N \times 1 \rightarrow N \times N \times 5 \rightarrow N \times N \times 10 \rightarrow N \times 1 \times 10 \rightarrow N \times N \times 10 \rightarrow N \times N \times 5 \rightarrow N \times N \times 1$

Discriminator: Given the graph size (number of nodes) N of a graph. The output feature map size of each layer through graph discriminator can be expressed as:

$N \times N \times 1 \rightarrow N \times N \times 5 \rightarrow N \times N \times 10 \rightarrow N \times 1 \times 10 \rightarrow 1 \times 1 \times 10$

For the edge to edge layers, the size of two kernels in two directions are $N \times 1$ and $1 \times N$. For the node to edge layer, the kernel size is $1 \times N$

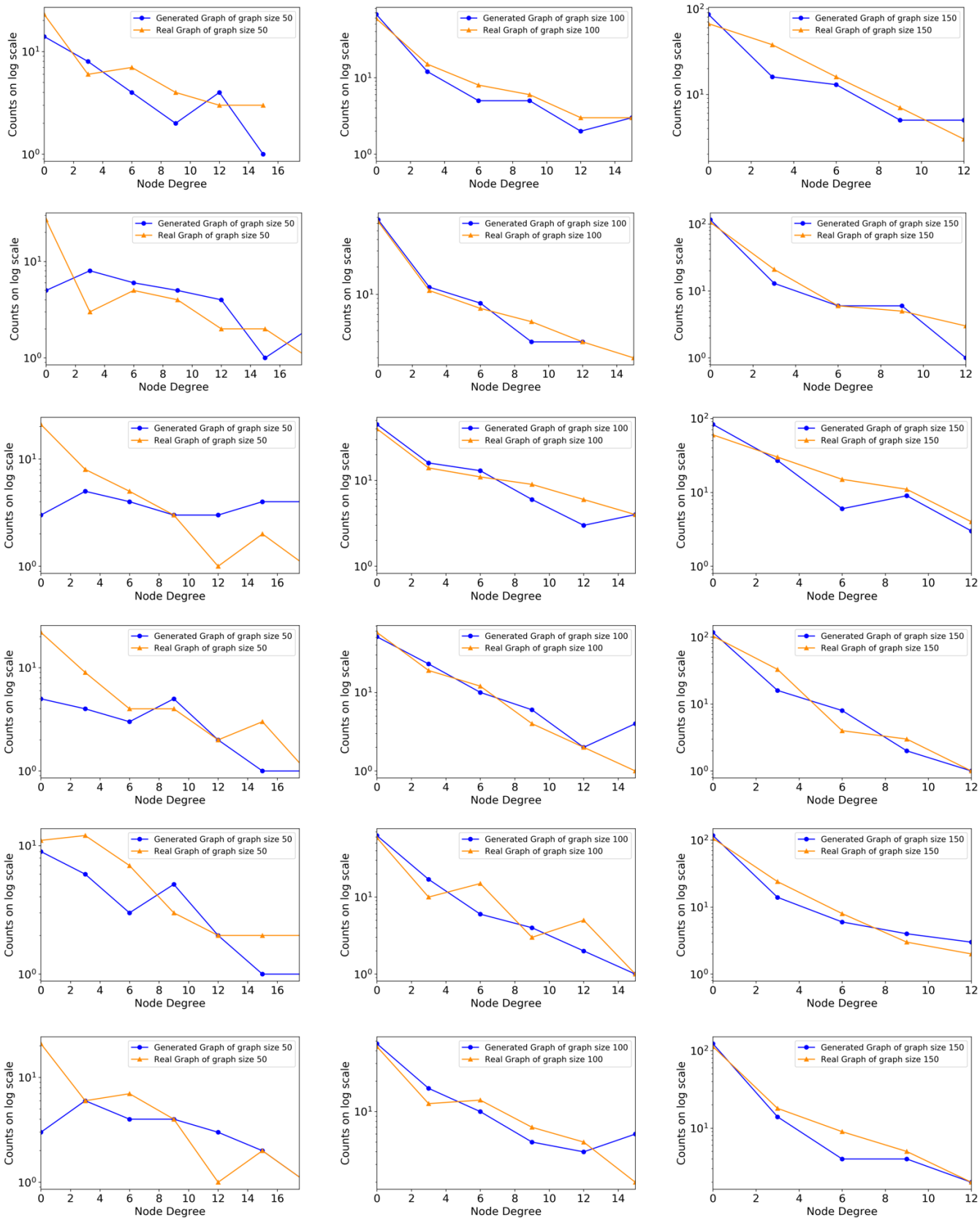

Figure 9: Examples of node degree distrbution for generated graphs and real graphs