Impact of Graph-to-Sequence Conversion Methods on the Accuracy of Graph Generation for Network Simulations

1st Kazuhiro Yasuda
Graduate School of Engineering
Nagaoka University of Technology
Nagaoka, Niigata, Japan
s213182@stn.nagaokaut.ac.jp

2nd Sho Tsugawa
Institute of Systems and Information Engineering
University of Tsukuba
Tsukuba, Ibaraki, Japan
s-tugawa@cs.tsukuba.ac.jp

3rd Kohei Watabe
Graduate School of Engineering
Nagaoka University of Technology
Nagaoka, Niigata, Japan
k_watabe@vos.nagaokaut.ac.jp

***Abstract*—In the field of communication network management, graph-based simulations using network topology models represented as graphs are widely adopted. In graph-based simulations, because there is a limited number of real network graph data that researchers and experts can access, the technique of generating graphs that mimic the features of real networks using graph generative models is essential. In this context, machine learning-based graph generative models have been rapidly advancing recently. In particular, in terms of the accuracy of reproducing the features of generated graphs, sequence data-based graph generative models have been successful. In this paper, we propose a method based on 2nd-order random walk as an alternative to DFS code, which is used for graph-to-sequence conversion in GraphTune, one of the sequence data-based graph generative models. We conducted experiments on a small dataset with limited diversity on a real graph dataset and confirmed that the model using the proposed method is at best 54.68% more accurate than the model using the conventional method.**

***Index Terms*—Graph-to-sequence, Graph generation, Conditional VAE, Graph feature, Generative model.**

I. Introduction

In the management and operation of communication networks, it is often necessary to prepare a network topology model represented as a graph for simulation and performance evaluation. Graph representation is a common method of representing network topology, and graph-based simulation is widely adopted. While it is desirable to use real network data as graph data, researchers and experts may not have access to graph data for all real networks. Moreover, available graph data (e.g., topological data held by telecommunication carriers) may not always have the number, scale, or features that are desirable for simulation or performance evaluation, mainly due to security issues. When it's not possible to obtain graph data from real networks that meet the desired conditions, experiments are conducted using artificially generated graph data created through graph generative models to simulate the features of real networks.

Historically, stochastic generative models utilizing predefined probabilities for edges and nodes have been explored as methods to artificially reproduce network topology structures. Various models, including Erdős-Rényi (ER) model [1], Watts-Strogatz (WS) model [2], and Barabási-Albert (BA) model [3], have been proposed [4]. However, many of these stochastic models focus solely on capturing specific single structural features of graphs, such as randomness [1], small worldness [2], scale-free features [3], and clustered nodes [5], with high accuracy.

On the other hand, machine learning-based graph generative models, which have rapidly advanced in recent years, aim to use machine learning techniques to learn the features of real networks and reproduce those features comprehensively across all aspects, thereby generating graphs with features more similar to those of real networks. These studies have succeeded in simultaneously reproducing various features reflecting the global structure of graphs, such as the average shortest path length, clustering coefficient, and power-law exponent of the degree distribution [4], [6]–[16].

Among machine learning-based graph generative models, sequence data-based models, including our previously proposed GraphTune [17], have achieved particular success in terms of feature reproduction accuracy [6]–[8], [13]. In these models, graphs are converted into sequential data and processed using recurrent neural networks to learn the graph's features. GraphTune is a conditional graph generative model that not only reproduces the features of the graphs used for training but also allows for the continuous tuning of generated graph features by specifying the value of features.

In previous evaluations of GraphTune, there has been no evaluation regarding the accuracy of the conversion method from graphs to sequential data. GraphTune uses Depth-First Search (DFS) code for the conversion from graphs to sequential data, demonstrating a high tuning accuracy in metrics such as the average shortest path length. However, it is unclear whether the DFS code can efficiently represent the structure of the graph because it cannot represent the relationships among neighboring nodes in a sequence because the order of node selection is lexicographic. Additionally, the conversion process to sequences lacks randomness, potentially leading to sparse sequence spaces and impacting the generation accuracy of graphs not in the training data.

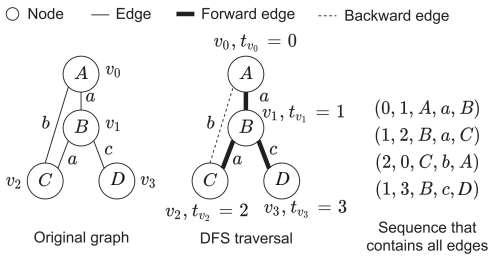


Fig. 1. Sequence of DFS code converted from an example graph.

In this paper, we propose a conversion method based on 2nd-order random walk instead of the DFS code-based graph-to-sequential data conversion employed in GraphTune. 2nd-order random walk is a technique that assigns weights to the selection probabilities of the next nodes in a random walk, incorporating both DFS-like and BFS (Breadth-First Search) strategies for node selection. 2nd-order random walk method, as adopted in node2vec [18], a representative model of node embedding, is known to effectively capture relationships with neighboring nodes in sequences based on its parameters. To the best of our knowledge, there have been no previous applications of 2nd-order random walk in the field of conditional graph generation, and the impact of the random walk property on the accuracy of graph generation has not yet been investigated.

The rest of this paper is structured as follows: In Section II, we present an overview of the GraphTune model. The proposed method is detailed in Section III. Section IV shows the evaluation experiments conducted for the proposed method. Finally, in Section V, we discuss the conclusions of this paper and outline future research directions.

II. GraphTune

GraphTune handles graph data as sequence data converted through a graph-to-sequence conversion method based on DFS code. The converted sequence data are processed by a Conditional Variational AutoEncoder (CVAE), which learns the reconstruction process of the input sequence. As a notational convention, a graph is represented by $G = (V, E)$, where V and E denote a subset of nodes $V \subseteq \{v_1, v_2, \ldots, v_n\}$ and a subset of edges $E \subseteq \{(x, y) \,|\, x, y \in V\}$, respectively.

A. DFS Code

In the conversion by DFS code, each node is given a timestamp by DFS, starting from 0. That is, each node v_i $(i = 0, 1, \ldots)$ receives a timestamp $t_{v_i} = (0, 1, \ldots, |V|-1)$ in the order it is discovered during the DFS. When a new timestamp is assigned during the DFS, traversed edges are stored in a sequence called the DFS traversal. For Fig. 1, the DFS traversal consists of $(v_0, v_1), (v_1, v_2), (v_1, v_3)$, and the node timestamps are assigned as follows: $t_{v_0} = 0$, $t_{v_1} = 1$, $t_{v_2} = 2$, $t_{v_3} = 3$. Edges included in the DFS traversal are called forward edges, while the rest are called backward edges. Based on the timestamps of nodes, an edge $e = (u, v)$ can be represented as $(t_u, t_v, L(u), L(e), L(v))$, where t_u and $L(\cdot)$ denote the timestamp of node u and the labels of nodes or edges, respectively.

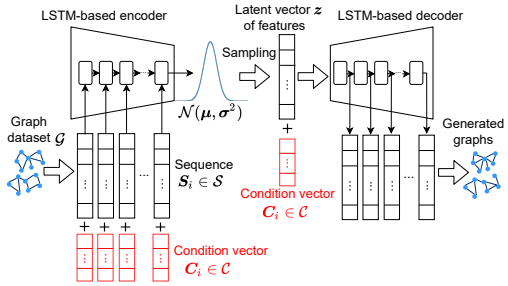


Fig. 2. The architecture of GraphTune [17].

Based on the order of graph traversal, DFS code constructs a sequence that includes all the edges within the graph. To construct a sequence that includes all the edges, backward edges (u, u') are inserted within the sequence generated by DFS code between the forward edges (w, u) and (u, v). If there are multiple backward edges such as (u, u') and (u, u''), the timestamps of u' and u'' are compared, and the one with the smaller timestamp is placed first. As a result, with an edge $e = (u, v)$, the graph is represented as a unique sequence of 5-tuples $(t_u, t_v, L(u), L(e), L(v))$.

B. Model Architecture

GraphTune consists of a CVAE with an LSTM-based encoder and decoder, as illustrated in Fig. 2. During training, the model takes a set of sequence data $\boldsymbol{S}_i = \boldsymbol{s}_0, \boldsymbol{s}_1, \ldots$ obtained by converting a graph dataset $\mathcal{G}$ into DFS codes as input. Simultaneously, it computes the desired graph features, which are provided to the model as a set of condition vectors $\mathcal{C}$. Following the architecture of a standard VAE, the encoder $F_{\text{enc}}(\boldsymbol{S}_i, \boldsymbol{C}_i)$ maps the sequence data $\boldsymbol{S}_i$ to parameters $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$. The latent vector $\boldsymbol{z}$ is sampled from a multivariate normal distribution, denoted as $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$. On the other hand, the decoder $F_{\text{dec}}(\boldsymbol{z}, \boldsymbol{C}_i)$ learns to reconstruct the original sequence $\boldsymbol{S}_i$, generating the reconstructed sequence $\tilde{\boldsymbol{S}}_i$.

During the generation process after training, we provide a condition vector $\boldsymbol{C}^*$ whose elements are feature values we intend to specify and a latent vector $\boldsymbol{z}$ randomly sampled from a multivariate normal distribution $\mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ to the decoder. The decoder then generates sequence data $\hat{\boldsymbol{S}}$ corresponding to a graph with features according to $\boldsymbol{C}^*$. The operation of the decoder can be summarized as follows:

$$\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}), \qquad \hat{\boldsymbol{S}} = F_{\text{dec}}(\boldsymbol{z}, \boldsymbol{C}^*). \tag{1}$$

III. Proposed Method

In this section, we explain a graph-to-sequence conversion method specifically designed for graph generation, utilizing 2nd-order random walks.

A. 2nd-Order Random Walk

The 2nd-order random walk is a graph sampling method initially proposed in node2vec [18], which is a representative model for node embeddings. It allows for flexible exploration compared with pure random walks by introducing bias to the

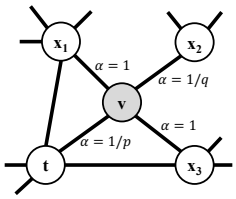


Fig. 3. Illustration of a step of a 2nd-order random walk.

random walk. In the 2nd-order random walk, two parameters, denoted as p and q, are used to weight the transition probabilities of nodes. Let us consider a random walk currently at node v which is about to traverse the edge (t, v) and move to another node (Fig. 3). Let the weight of edge (v, x) be w_{vx} (in an unweighted graph, $w_{vx} = 1$). The unnormalized transition probability from node v to another node x is expressed as $\pi_{vx} = \alpha_{pq}(t, x) \cdot w_{vx}$, where α_{pq} denotes the search bias for transition probabilities and is defined as follows:

$$\alpha_{pq} = \begin{cases} 1/p & \text{if } d_{tx} = 0, \\ 1 & \text{if } d_{tx} = 1, \\ 1/q & \text{if } d_{tx} = 2, \end{cases} \tag{2}$$

where d_{tx} denotes the shortest path length between nodes t and x, with a value of one among $0, 1, 2$.

The parameters p and q in the 2nd-order random walk are referred to as the return (inward) hyper-parameter and in-out hyper-parameter, respectively, which control the order of node exploration. p governs the likelihood of revisiting the node visited just before, while q regulates the probability of exploring more distant parts of the graph.

- *BFS-like conversion*: By setting $p < \min(1, q)$, the 2nd-order random walk is encouraged to take a step back, promoting the tendency to backtrack, which helps maintain a localized walk by returning to the source node t.
- *DFS-like conversion*: By setting $q < \min(1, p)$, the 2nd-order random walk is encouraged to head toward more distant nodes, promoting the tendency to explore further, which enables the capture of the homophily of nodes (i.e., the property that nodes belonging to similar communities are embedded in close vectors in the feature space).

B. Graph-to-Sequence Conversion

In our proposed method, we suggest using a 2nd-order random walk as an alternative to the DFS code employed in the graph-to-sequence conversion, as done in GraphTune. More specifically, this is achieved by replacing the conversion through DFS, as described in Section II-A, with the 2nd-order random walk explained in Section III-A. To evaluate the impact of the graph-to-sequence conversion method, the format of the data obtained through the conversion remains the same as the 5-tuples $(t_u, t_v, L(u), L(e), L(v))$ described in Section II-A. Thus, the assumptions used in this paper are the same as those in the original GraphTune paper [17], except for those indicated in this paper.

One of the anticipated advantages of adopting the 2nd-order random walk is the potential utilization of structural features of graphs that were not captured by the DFS-based conversion. In GraphTune [17], the use of DFS code follows GraphGen [13], but the superiority of DFS in the graph-to-sequence conversion has not been sufficiently validated. By using the 2nd-order random walk, it is expected that by adjusting the parameters, a sequence representation that combines BFS-like and DFS-like characteristics can be obtained.

Another anticipated advantage of using the 2nd-order random walk is the introduction of randomness in the conversion to sequence data. While the sequence data obtained through DFS code is unique for a given graph, our proposed method uses the 2nd-order random walk, which results in sequence data that includes randomness for a single graph. As a result, this reduces the sparsity of data in the space of converted sequence data and increases the diversity of sequence data converted from the graph. Consequently, it is expected to enable robust learning for small datasets with limited data diversity.

IV. Experiments

Using the graph-to-sequence conversion with the 2nd-order random walk, we train a conditional graph generative model on real graphs and compare the results with the conventional DFS code-based learning. This allows us to investigate the impact of the randomness in the graph-to-sequence conversion on the accuracy of conditional graph generation. In this section, we adopt GraphTune [17] as a graph generative model and validate the accuracy using the proposed conversion method from graphs to sequence data. Through experimental results, we demonstrate that our approach outperforms the conventional DFS code, especially for a small datasets.

A. Dataset

To evaluate the performance of graph-to-sequence conversion methods on real graphs, we use the same graph dataset as the original GraphTune [17] for evaluation. The graphs in this dataset are sampled from the Twitter "who-follows-whom" graph of the Higgs Twitter Dataset [19]. Detailed information about the sampling method is provided in the original GraphTune paper. We divide this dataset into two subsets: the training and the validation sets. In the original GraphTune paper, the size ratio between the training and the validation set is 90% and 10%, respectively, for a total of 2000 graphs. However, in this study, to specifically assess the robustness for a small dataset, we configure both the training and validation sets to 10%.

B. Parameter Settings

The parameters p and q of the 2nd-order random walk for our proposed method are varied within the range of $[0.25, 0.50, 1.00, 2.00, 4.00]$, and the performance is evaluated for all combinations of p and q. To mitigate the impact of random effects during dataset creation and training, models are trained 10 times for each combination of p and q values. The evaluation is then conducted on the basis of the average and standard deviation of accuracy over these 10 trained models.

In addition, the structural features given as conditions for the graphs generated by GraphTune are specified as the average shortest path length. The specified condition is to generate graphs with average shortest path lengths of 3, 4, and 5. A total of 3,300 graphs are generated for each specified value, and the generated graphs are evaluated for errors from the specified conditions. On the other hand, for the hyperparameters of GraphTune other than those mentioned above, those recommended in the original GraphTune paper [17] are used.

C. Performance Metrics

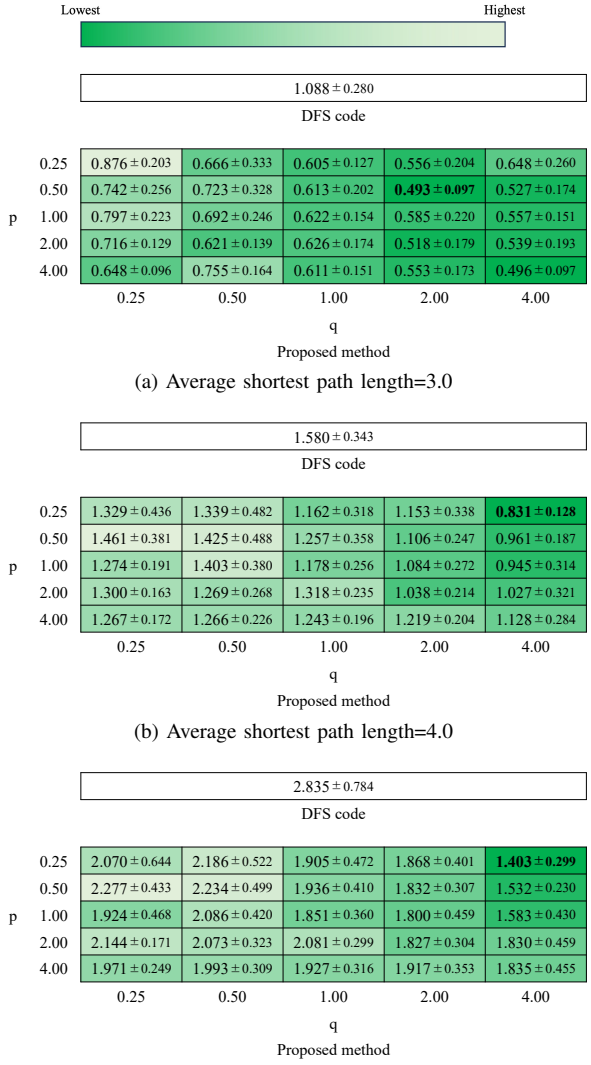
Since GraphTune is a generative model that allows features to be specified depending on condition, its performance can be measured by the accuracy of the specification by condition. Therefore, the performance metric used for evaluation is the Root Mean Squared Error (RMSE) between the specified values and the feature values of the generated graphs (i.e., average of shortest path length). The average and standard deviation of RMSE are calculated for each set of parameters p and q for the 2nd-order random walk and used for evaluation.

D. Performance Evaluation

In this section, we evaluate the specified accuracy of the structural features of the graphs generated by GraphTune using the proposed method with the RMSE metric, as described in Section IV-C.

The RMSE values for models using the conventional DFS code and those using the proposed method are shown in Fig.4. Bold values in Fig. 4 indicate that these values represent the best average values in the figure. As observed in Fig. 4, the RMSE values are consistently smaller for the proposed method in all corresponding cases, demonstrating the ability of the proposed method to accurately specify the features of the generated graphs. The improvement in RMSE shows a maximum decrease of up to 54.68%, 47.41%, and 50.51%, respectively. Additionally, focusing on the combinations of p and q in Fig. 4, when $q > 1 > p$, indicating a more "BFS-like" parameters, the results tend to be more accurate.

The above results are from the small dataset, which has a smaller number of graphs than the dataset used in the original GraphTune paper. When using a dataset of the same size as that in the paper, the improvement in accuracy was not significant compared with the results above. This suggests that the introduced randomness in the proposed method effectively mitigates the sparsity issue in the sequence space converted from graphs, which is especially noticeable in small datasets, and enhances the accuracy of the generative model.



(a) Average shortest path length=3.0

(b) Average shortest path length=4.0

(c) Average shortest path length=5.0

Fig. 4. RMSE values for GraphTune models using DFS code and heatmaps of RMSE for the models using the proposed method.

V. Conclusions

In this paper, we replaced the conventional DFS-based method with the 2nd-order random walk-based method for graph-to-sequence conversion in the sequence-based graph generative model, GraphTune, and evaluated the generation accuracy. In the evaluation process, we used real network datasets and evaluated the accuracy of specifying the features of the generated graphs using GraphTune. From the results of the evaluation, it was confirmed that a significant improvement in accuracy, particularly for small datasets with a limited number of graphs, was observed. We observed a maximum improvement of 54.68% in accuracy compared with the conventional graph-to-sequence conversion method, DFS code. The results of this study suggest that the randomness introduced during the conversion to sequences can mitigate the sparsity issues in the data space. Future challenges includes validation of the proposed method on more datasets of different sizes and domains, validation for more sequence-based models other than GraphTune, and the introduction of new evaluation metrics.

Acknowledgments

This work was partly supported by JSPS KAKENHI Grant Number JP23H03379.

REFERENCES

[1] P. Erdös and A. Rényi, "On Random Graphs I," *Publicationes Mathematicae*, vol. 6, no. 26, pp. 290–297, 1959.

[2] D. J. Watts and S. H. Strogatz, "Collective Dynamics of 'Small-World' Networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[3] R. Albert and A.-L. Barabási, "Statistical Mechanics of Complex Networks," *Reviews of Modern Physics*, vol. 74, no. 1, pp. 47–97, 2002.

[4] A. Bonifati, I. Holubová, A. Prat-Pérez, and S. Sakr, "Graph Generators: State of the Art and Open Challenges," *ACM Computing Surveys*, vol. 53, no. 2, 2021.

[5] P. W. Holland, K. B. Laskey, and S. Leinhardt, "Stochastic Blockmodels: First Steps," *Social Networks*, vol. 5, no. 2, pp. 109–137, 1983.

[6] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec, "GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models," in *Proc. of the 35th International Conference on Machine Learning (ICML 2018)*, 2018.

[7] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia, "Learning Deep Generative Model of Graphs," in *Proc. of 6th International Conference on Learning Representations (ICLR 2018) Workshop*, 2018.

[8] A. Bojchevski, O. Shchur, D. Zügner, and S. Günnemann, "NetGAN: Generating Graphs via Random Walks," in *Proc. of the 35th International Conference on Machine Learning (ICML 2018)*, 2018, pp. 609–618.

[9] M. Simonovsky and N. Komodakis, "GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders," in *Proc. of the 27th International Conference on Artificial Neural Networks (ICANN 2018)*, 2018.

[10] R. Assouel, M. Ahmed, M. H. Segler, A. Saffari, and Y. Bengio, "DEFactor: Differentiable Edge Factorization-Based Probabilistic Graph Generation," *arXiv*, 2018.

[11] C. Yang, P. Zhuang, W. Shi, A. Luu, and P. Li, "Conditional Structure Generation through Graph Variational Generative Adversarial Nets," in *Proc. of the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*, 2019, pp. 1338–1349.

[12] J. Lim, S.-Y. Hwang, S. Moon, S. Kim, and W. Y. Kim, "Scaffold-Based Molecular Design with a Graph Generative Model," *Chemical Science*, vol. 2020, no. 4, pp. 1153–1164, 2020.

[13] N. Goyal, H. V. Jain, and S. Ranu, "GraphGen: A Scalable Approach to Domain-agnostic Labeled Graph Generation," in *Proc. of the Web Conference 2020 (WWW 2020)*, 2020, pp. 1253–1263.

[14] W. Jin, R. Barzilay, and T. Jaakkola, "Hierarchical Generation of Molecular Graphs using Structural Motifs," in *Proc. of the 37th International Conference on Machine Learning (ICML 2020)*, 2020.

[15] X. Guo and L. Zhao, "A Systematic Survey on Deep Generative Models for Graph Generation," *arXiv*, 2020.

[16] F. Faez, Y. Ommi, M. S. Baghshah, and H. R. Rabiee, "Deep Graph Generators: A Survey," *IEEE Access*, vol. 9, pp. 106 675–106 702, 2021.

[17] K. Watabe, S. Nakazawa, Y. Sato, S. Tsugawa, and K. Nakagawa, "GraphTune: A Learning-Based Graph Generative Model With Tunable Structural Features," *IEEE Transactions on Network Science and Engineering*, vol. 10, no. 4, pp. 2226–2238, 2023.

[18] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proc. of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD 2016)*, 2016, pp. 855–864.

[19] M. D. Domenico, A. Lima, P. Mougel, and M. Musolesi, "The Anatomy of a Scientific Rumor," *Scientific Reports*, vol. 3, no. 2980, 2013.