GraphTune: A Learning-based Graph Generative Model with Tunable Structural Features

Kohei Watabe, *Member, IEEE*, Shohei Nakazawa, Yoshiki Sato, Sho Tsugawa, *Member, IEEE,* and Kenji Nakagawa

Abstract—Generative models for graphs have been actively studied for decades, and they have a wide range of applications. Recently, learning-based graph generation that reproduces real-world graphs has been attracting the attention of many researchers. Although several generative models that utilize modern machine learning technologies have been proposed, conditional generation of general graphs has been less explored in the field. In this paper, we propose a generative model that allows us to tune the value of a global-level structural feature as a condition. Our model, called GraphTune, makes it possible to tune the value of any structural feature of generated graphs using Long Short Term Memory (LSTM) and a Conditional Variational AutoEncoder (CVAE). We performed comparative evaluations of GraphTune and conventional models on a real graph dataset. The evaluations show that GraphTune makes it possible to more clearly tune the value of a global-level structural feature better than conventional models.

Index Terms—Graph generation, Conditional VAE, LSTM, Graph feature, Generative model.

I. INTRODUCTION

Generative models for graphs have a wide range of applications, including communication networks, social networks, transportation systems, databases, cheminformatics, and epidemics. Repeated simulation on graphs is a basic approach to discovering information in the above fields of study. However, researchers and practitioners do not always have access to enough real graph data. Generative models of graphs can supplement a graph dataset that does not include a sufficient number of real graphs. Moreover, generating graphs that are not included in a dataset or future graphs can be used to discover novel synthesizable molecules [2]–[4] or to predict the growth of a network [5].

Classically, stochastic models that generate graphs with a pre-defined probability of edges and nodes have been studied, and they focus on only a single-aspect feature of graphs. Various models [6] have been proposed in the literature,

K. Watabe, Y. Sato, and K. Nakagawa are with the Graduate School of Engineering Nagaoka University of Technology, Nagaoka, Niigata, Japan (e-mail: k_watabe@vos.nagaokaut.ac.jp; s171039@stn.nagaokaut.ac.jp; nakagawa@nagaokaut.ac.jp).

S. Nakazawa was with the Graduate School of Engineering Nagaoka University of Technology, Nagaoka, Niigata, Japan.

S. Tsugawa is with the Faculty of Engineering, Information and Systems University of Tsukuba, Tsukuba, Ibaraki, Japan (e-mail: stugawa@cs.tsukuba.ac.jp).

including the Erdős-Rényi (ER) model [7], Watts-Strogatz (WS) model [8], and Barabási-Albert (BA) model [9]. These stochastic models accurately reproduce a specific target structural feature (e.g., randomness [7], small worldness [8], scale-free features [9], and clustered nodes [10]). In other words, they cannot be adapted to the real data of graphs with numerous features, and cannot guarantee that generated graphs completely reproduce all features of the graphs excluding the target feature.

Generative models for graphs using machine learning technology learn features from graph data and try to reproduce those features according to the data in every single aspect [5], [6], [11]–[20]. For the last several years, learning-based graph generation has been attracting the attention of many researchers, and several approaches have been tried in recent studies. Although a lot of models have been proposed to generate small graphs with the aim of designing molecules [3], [4], [14], [16], [18], some recent studies [5], [17] enable the generation of relatively large graphs that also include citation graphs and social networks. In particular, the sequence data-based approach, which converts a graph into sequential data and learns the sequential data by recurrent neural networks, has been successful in the field [5], [11], [12], [17]. These studies reproduce various features that reflect the global structures of graphs, including average shortest path length, clustering coefficient, and the power-law exponent of the degree distribution.

Although the existing generative models for graphs using machine learning technology can generate similar graphs to real-world graphs, most of them cannot generate graphs that have user-specified structural features. Although demand for conditional generation of graphs with specific features is common, it has been less explored [19], [20].

Despite the fact that some works aim for conditional generation of graphs with a specific feature, their applicability and performance are not sufficient to generate general graphs with a specific value of a structural feature. Several models [14], [16], [18] that enable conditional generation utilize domain-specific knowledge of molecule chemistry and are not suitable for graphs of other domains. DeepGMG [12], one of the pioneering studies of conditional graph generation, does not assume domain-specific knowledge explicitly, and can generate graphs according to a specific condition. However, the work just evaluates generation conditioned by the number of atoms (nodes), bonds (edges), or aromatic rings (hexagons) in a molecule. DeepGMG does not have an ability to tune global-level structural features (e.g., average shortest path length, clustering coefficient, and the power-law exponent of

the degree distribution) which are difficult to tune by adding or removing a local structure of a graph such as a node, edge, or hexagon. Although attempts have been made to train DeepGMG by graphs generated by the BA model, they have succeeded in unconditional generation of only very small graphs with 15 nodes [12]. CondGen [15], whose applicable domain is not limited to molecule chemistry, has achieved conditional generation of general graphs including citation graphs. CondGen can reproduce global-level structural features (average shortest path length and Gini index are evaluated in the paper [15]). CondGen succeeds in improving generation by using datasets grouped by label by inputting labels as a condition. Unfortunately, it does not provide a model to continuously tune a feature since it requires training datasets grouped by labels. It does not have sufficient performance to tune the features flexibly based on conditions given as a continuous value of a global-level feature (we discuss the performance of CondGen in Section VI).

In this paper, we propose GraphTune, a graph generative model that makes it possible to tune the value of any structural feature of a generated graph using Long Short Term Memory (LSTM) [21] and a Conditional AutoEncoder (CVAE) [22]. GraphTune adopts a sequence data-based approach for learning graph structures, and graphs are converted to sequence data by using Depth-First Search (DFS) code that achieves success in GraphGen [17]. Unlike GraphGen, GraphTune is a CVAE-based model and the CVAE in the model is composed of an LSTM-based encoder and decoder. GraphTune uses CVAE to generate a graph with some specific feature, including global-level structural features, and the feature can be continuously tuned. Meanwhile, features other than the specified feature are accurately reproduced in every single aspect according to the dataset that is learned.

In summary, the main contributions of this paper are as follows:

- We propose a novel learning-based graph generative model called GraphTune with tunable structural features. In GraphTune, flexible generation of graphs with any feature including global-level structural features (e.g., average shortest path length and clustering coefficient) can be achieved by giving the value of a feature as a condition vector to the CVAE-based architecture.
- We achieve elaborate reproduction of a graph dataset in every single aspect by adopting a sequence data-based approach for learning. GraphTune tunes the value of a specific feature to a specified value while keeping values of other features within the range of values that exist in the dataset.
- We perform empirical evaluations of GraphTune on a real graph dataset. The evaluation results establish that the tunability and reproducibility of graphs in GraphTune outperforms those in conventional conditional and unconditional graph generative models. The code and dataset used for empirical evaluations are available on GitHub[1].

The rest of this paper is structured as follows. In Section II, we summarize related works in the field of graph generative

models. Section III formulates the generation problem of a graph with a specified feature. In Section IV, we introduce the DFS code that we adopt as a method for converting graphs into sequence data in our model. We explain the model architecture and the training and generation algorithms of GraphTune in Section V. Section VI shows the empirical evaluations of GraphTune and conventional models. Section VII discusses the limitations of the paper and future research directions. Finally, Section VIII concludes the paper.

II. RELATED WORKS

Graph generation has a long history and the literature is rich with the results of many researchers. One of the most rudimentary models, Erdős-Rényi model, generates simple random graphs and was proposed in 1959. Around 2000, two models [8], [9] that reproduce structural features of graphs called small-world networks and power-law degree distribution attracted the attention of researchers. Since these studies were reported, various statistical graph generation methods inspired by them have been proposed [6], [23]–[25]. A lot of the structural features of graphs have also been quantified in this research. What can be said in common with these traditional statistical generation models is that a model focuses on one (or a few) of the many features and aims to reproduce the features (e.g., small worldness, power-law degree distribution, and local clustering). These models cannot be adapted to real graph datasets that have numerous features, and cannot guarantee that the generated graphs completely reproduce every single aspect according to the real data.

A recent trend in the field of graph generation is learning-based models that reproduce real-world graphs. Learning-based models have evolved rapidly over the last few years, and have attracted the attention of researchers. Learning-based models have been proposed for a wide range of domains ranging from discovering new molecular structures to modeling social networks, and most recently, survey papers have been published [6], [19], [20]. Various learning-based models have been proposed, including adjacency-based and edge-list-based approaches. The sequence data-based approach that converts a graph into sequential data has been particuarly successful in this field [5], [11], [12], [17].

Although several learning-based models for graphs have been proposed, conditional generation of graphs is less explored [19], [20]. The few models that achieve conditional generation of graphs include domain-specific models in the field of molecule chemistry [3], [4], evolutionary developmental biology [26], and natural language processing [27], [28]. Unfortunately, these models cannot easily be applied to general graphs due to the utilization of domain-specific knowledge. DeepGMG [12] and CondGen [15] have been proposed as models applicable to general graphs. DeepGMG adds a condition into the latent vector during the decoding process of the graph to tune the local structure in the graph such as the number of nodes, edges, or hexagons. CondGen allows tuning of global-level structural features (including average shortest path length and Gini index at least) with graph variational generative adversarial nets. TSGG-GAN also
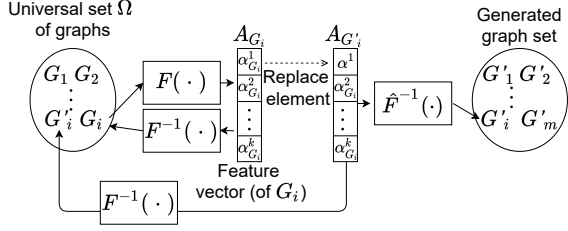
---

[1]https://github.com/wkouw1082/GraphTune

Fig. 1. Problem formulation. A graph $G_i \in \Omega$ is mapped to a feature vector $A_{G_i} \in A$ by a mapping $F(G_i) = A_{G_i} = [\alpha_{G_i}^1 \alpha_{G_i}^2 \cdots]^T$. Elements of a feature vector A_{G_i} for graph G_i represent values of all sorts of features that are calculated from G_i. This paper tackles the inference problem of finding $\hat{F}^{-1}(\cdot)$ that approximates $F^{-1}(\cdot)$, using a subset $\mathcal{G}$ of the universal set Ω of graphs. $\hat{F}^{-1}(\cdot)$ can generate G_i' by inputting the feature vector $A_{G_i'}$ in which any element of the vector A_{G_i} is replaced by an arbitrary value.

Fig. 2. Sequence converted from an example graph. All nodes $v_i (i = 0, 1, \dots)$ are assigned timestamps t_{v_i} in order of depth-first search, thereby obtaining a DFS traversal. The edges contained in the DFS traversal are called forward edges, and other edges are called backward edges. A backward edge (u, u') is placed between forward edges (w, u) and (u, v), thereby obtaining a sequence that contains all edges.

provides conditional generation of graphs, but it focuses on time series conditioned generation and the challenges are different from ours. In TSGG-GAN, multivariate time series data are input as node expression values to the model, and graphs conditioned by the time series are generated.

III. PROBLEM FORMULATION

The graphs treated in this paper are undirected connected graphs without self-loop. As a notational convention, a graph is represented by $G = (V, E)$, where V and E denote a subset of nodes $V \subseteq \{v_1, v_2, \dots, v_n\}$ and a subset of edges $E \subseteq \{(x, y) \mid x, y \in V\}$, respectively. We let $\Omega = \{G_1, G_2, \dots\}$ denote the universal set of graphs $G = (V, E)$.

We consider a mapping $F : \Omega \to A$, and a graph $G_i \in \Omega$ that is mapped to a feature vector $A_{G_i} \in A$ by $F(G_i) = A_{G_i} = [\alpha_{G_i}^1 \alpha_{G_i}^2 \cdots]^T$ as shown in Fig. 1. The jth element $\alpha_{G_i}^j$ of the vector A_{G_i} expresses a feature of graph G_i. Every feature is represented by a real number $\alpha_{G_i}^k \in \mathbb{R}$. For example, elements of A_{G_i} represent values of features such as the number of nodes and edges, average shortest path length, average degree, edge density, modularity, clustering coefficient, power-law exponent of the degree distribution, and largest component size.

In this paper, we formulate the problem of generating a graph with specified features as inferencing the inverse image from feature vectors of graphs to graphs (see Fig.1). We define the inverse image $F^{-1}(\cdot)$ of the mapping $F(\cdot)$, and graph G_i can be obtained by calculating $F^{-1}(A_{G_i}) = \{G_i, \dots\}$ with the inverse image. We tackle the inferencing problem of finding $\hat{F}^{-1}(\cdot)$ that approximates $F^{-1}(\cdot)$ by using a subset $\mathcal{G}$ of Ω. By solving this problem, it is possible to generate G_i' with the feature vector $A_{G_i'}$ in which any element of the vector A_{G_i} is replaced by an arbitrary value. Since we cannot use the universal set Ω of graphs, the inference needs to be achieved by a subset $\mathcal{G}$ of Ω that is contained in an accessible dataset.

IV. DFS CODE

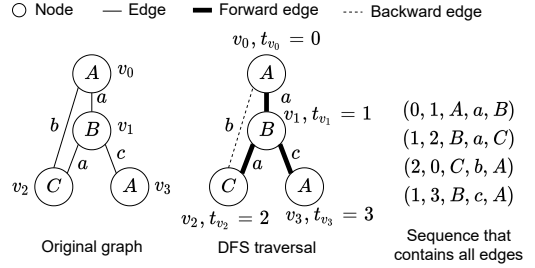GraphGen [17] is a successful model for unconditional generation in learning-based graph generation that utilizes the DFS code. The key idea is to use the DFS code to convert graphs into sequence data. The converted sequence data are learned by LSTM in a training process. The compact expression of a graph by sequence data with DFS code allows GraphGen to accurately generate graphs that are similar to graphs in a dataset. While we propose a novel CVAE-based generative model for graphs different from GraphGen, our model follows the sequence data-based approach by using DFS code in the preprocess for training. In this section, we summarize the conversion process by DFS code that is common to GraphGen and our model.

DFS code converts a graph to a unique sequence of edges that retains the structural features of the graph. It is well known that a trajectory (i.e., a sequence) of a walk on a graph reflects features of the graph, including the degree distribution [29]. DFS code converts a graph to a compact sequence of length $|E|$ by using the depth-first search preventing revisit of edges.

In the conversion algorithm of DFS code, timestamps are first added to all nodes from 0 by performing the depth-first search. That is, all nodes $v_i (i = 0, 1, \dots)$ are discovered and assigned timestamps t_{v_i} in order of depth-first search. The traversal of the search is represented as a sequence of edges called a DFS traversal. In the example graph shown in Fig. 2, a DFS traversal is $(v_0, v_1), (v_1, v_2), (v_1, v_3)$, and timestamps of nodes are assigned as $t_{v_0} = 0$, $t_{v_1} = 1$, $t_{v_2} = 2$, and $t_{v_3} = 3$. Edges contained in a DFS traversal are called forward edges, and other edges are called backward edges. By adding a timestamp to all nodes, edge $e = (u, v)$ can be annotated as a 5-tuple $(t_u, t_v, L(u), L(e), L(v))$, where t_u and $L(\cdot)$ denote the timestamp of node u and the label of the edge or node, respectively. Although the graphs treated in this paper are graphs with unlabeled nodes and edges, the original DFS code is designed for labeled graphs. A detailed treatment of $L(u)$, $L(e)$ in our model is explained in Section V-A.

Based on the order of node timestamps, DFS code constructs a sequence that contains all edges in a graph. Although the forward edges are already constructed as a sequence (i.e., a DFS traversal), the backward edges are not included in the sequence. To construct a sequence that contains all edges in a graph, a backward edge (u, u') is placed between forward edges (w, u) and (u, v). If there are multiple backward edges (u, u') and (u, u''), the timestamps of u' and u'' are compared, and the smaller one is placed in front. By performing this

procedure, all backward edges are placed between forward edges, and a sequence that contains all edges is obtained. In the example shown in Fig. 2, the obtained sequence is $(0, 1, A, a, B), (1, 2, B, a, C), (2, 0, C, b, A), (1, 3, B, c, A)$. Although sequences that are constructed by the above procedure are not necessarily unique, the lexicographically smallest sequence based upon lexicographical ordering [30] is chosen as the unique one. As a result, the graph is represented as unique sequence of 5-tuple $(t_u, t_v, L(u), L(e), L(v))$ by DFS code.

It is not difficult to reconvert a DFS code into a graph if the DFS code exactly represents the graph. By constructing the edges and the nodes on either side of these edges in the order listed in the DFS code, we can reconstruct the graph. In the case of the example shown in Fig 2, the graph can be reconstructed by the following procedure. 1) Create Node 0, Node 1, and Edge 0-1 with labels A, B, and a respectively; 2) Create new Node 2 and Edge 1-2 with labels C and a, respectively; 3) Create new Edge 2-0 with label b. 4) Create new Node 3 and Edge 1-3 with labels A and c, respectively.

V. GRAPHTUNE: A GRAPH GENERATIVE MODEL WITH TUNABLE STRUCTURAL FEATURES

We propose *GraphTune* – a generative model for graphs that is able to tune a specific structural feature using DFS code and CVAE. GraphTune is composed of CVAE with an LSTM-based encoder and decoder. Graphs are converted to sequence data by the DFS code, and the sequence data are input to LSTM. This section provides a detailed review of generative approaches of graphs in GraphTune.

A. Sequence Data Converted from Graphs

Like GraphGen, GraphTune learns a sequence dataset $\mathcal{S} = \{F_{\text{DFS}}(G_i) | G_i \in \mathcal{G}\}$ that is converted from a graph dataset $\mathcal{G} \subset \Omega$ using a conversion $F_{\text{DFS}}(\cdot)$ based on DFS code. Although the conversion $F_{\text{DFS}}(\cdot)$ is basically the same as the conversion by DFS code described in Section IV, it is modified as follows to adapt it for our problem. As mentioned above, although the original DFS code is designed for labeled graphs, we assume unlabeled graphs in Section III. In the sequence dataset that is learned by GraphTune, node labels $L(v)$ ($v \in V$) and edge labels $L(e)$ ($e \in E$) in a 5-tuple are set as the degree of node v and 0, respectively. According to the DFS code procedure described in Section IV, a graph is converted to a sequence of 5-tuples. At the end of the sequences, an End Of Sequence (EOS) token $(\text{EOS}_t, \text{EOS}_t, \text{EOS}_L, 1, \text{EOS}_L)$ is added, where EOS_t and EOS_L represent $1 + \max_{v \in V} t_v$ and $1 + \max_{v \in V} L(v)$ for a set V of all nodes in the graph dataset $\mathcal{G}$. Sequences with EOS allow us to learn graphs of any size. We can obtain a sequence $\mathbf{S}_i \in \mathcal{S}$ by further converting the sequence of 5-tuples by component-wise one-hot encoding. A sequence $\mathbf{S}_i$ with element $\mathbf{s}_j \in \{0, 1\}^k$ is input into the model of GraphTune (see below for the model architecture).

B. Condition Vectors Corresponding to Graphs

Along with the sequential data fed by the DFS code, we input condition vectors $\mathcal{C}$ expressing structural features of
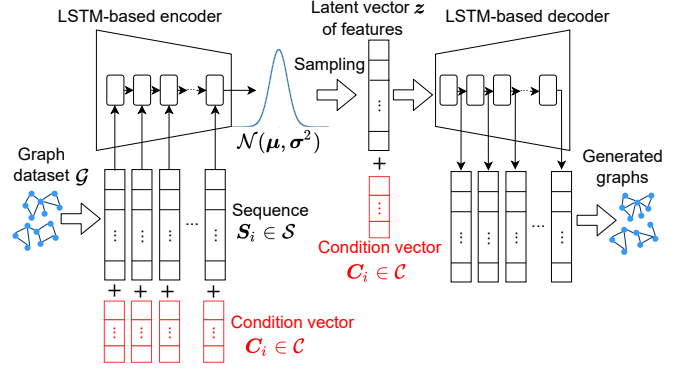

Fig. 3. Proposed model composed of CVAE with a LSTM-based encoder and decoder. A graph in the graph dataset $\mathcal{G}$ is converted to a sequence $S_i \in \mathcal{S}$ by using DFS code. The sequence is processed by an LSTM-based encoder. The decoder generates a sequence of 5-tuples, and the sequence is converted to a generated graph. The condition vector is input to both the encoder and decoder. See the equations in Section V-C for the detailed process of the model.

the graph dataset $\mathcal{G}$ to the GraphTune model for learning. Elements c_i of vector $\mathbf{C}_i = [c_i, c_i, \ldots, c_i]^T \in \mathcal{C}$ represent the value of the structural feature of graph G_i in graph dataset $\mathcal{G}$, and c_i corresponds to the first element of a feature vector A_{G_i} in Fig.1. Elements c_i of condition vector $\mathbf{C}_i$ are calculated from graph G_i by a statistical process. The condition vector specifies the structural features we focus on, and we can choose any structural feature as the elements of a condition vector. Structural features that can be specified in a condition vector are not limited to features regarding the local structures of graphs such as the number of nodes and edges, but global-level structural features (including average of shortest path length, clustering coefficient and the power-law exponent of the degree distribution) can also be specified. For example, if we want to tune the model by focusing on the clustering coefficient of the graphs, then we calculate a clustering coefficient $\alpha_{\text{cls}}(G_i)$ for each graph $G_i \in \mathcal{G}$ and construct a vector $[\alpha_{\text{cls}}(G_i), \alpha_{\text{cls}}(G_i), \ldots, \alpha_{\text{cls}}(G_i)]$ for each $G_i \in \mathcal{G}$.

C. Model Architecture

The proposed model is composed of CVAE with an LSTM-based encoder and decoder (see Fig. 3). A graph dataset $\mathcal{G}$ is converted to a sequence dataset $\mathcal{S}$ in the manner explained in Section IV. By doing this, set of condition vectors $\mathcal{C}$ is calculated from the graph dataset $\mathcal{G}$. The proposed model is trained with a sequence dataset $\mathcal{S}$ and a condition vector set $\mathcal{C}$. Sequence data $\mathbf{S}_i \in \mathcal{S}$ and the condition vector $\mathbf{C}_i \in \mathcal{C}$ are input into the LSTM-based encoder, and the encoder finds a latent state distribution $\mathcal{Z}$ of latent vectors. A latent vector z is randomly sampled from the distribution $\mathcal{Z}$, and the latent vector z concatenated with a condition vector $\mathbf{C}_i \in \mathcal{C}$ is input into the LSTM-based decoder. The decoder tries to reproduce the sequence $\mathbf{S}_i$. The details of the encoder and the decoder are described below.

Encoder: Encoder $q_\phi(z|\mathbf{S}_i)$ with parameter ϕ learns sequences $\mathbf{S}_i$ and maps them to a latent vector z according to

features of graphs. To treat sequence data, we employ a stacked LSTM as an encoder. To encode a graph into a latent space, the jth element s_j of the sequence S_i and a condition vector C_i are vertically concatenated in a single vector and the vector is embedded with a single fully connected layer f_{eemb}. The embedded vector is then fed into each LSTM block f_{enc}. The initial hidden state vector h_0 is initialized as a zero vector $\mathbf{0}$. The stacked LSTM with the embedding layer f_{eemb} processes a sequence S_i of length $k = |S_i|$ by recursively applying the LSTM block f_{enc} to hidden state vector h_j. The output h_k of the last LSTM block is fed to two functions f_μ and f_{σ^2} implemented by single fully connected layers. As usual in VAE, the latent state distribution is enforced as a multivariate Gaussian distribution with dimension L. A latent vector z is then sampled from the latent state distribution $\mathcal{N}(\mu, \sigma^2)$ where $\mu = f_\mu(h_k)$ and $\sigma^2 = f_{\sigma^2}(h_k)$. Summarizing the above, the process of the encoder part of our model is as follows:

$$h_0 = \mathbf{0}, \tag{1}$$

$$h_{j+1} = f_{\text{enc}}(h_j, f_{\text{eemb}}([s_j^T, C_i^T]^T)) \quad (j = 0, 1, \ldots, k-1), \tag{2}$$

$$\mu = f_\mu(h_k) \tag{3}$$

$$\sigma^2 = f_{\sigma^2}(h_k) \tag{4}$$

$$z \sim \mathcal{N}(\mu, \sigma^2). \tag{5}$$

Decoder: Decoder $p_\theta(S_i|z)$ with parameter θ learns to map a subsequence $\{s_m | m \le j\}$ of a sequence S_i, a condition vector C_i, and latent vector z to a next element s_{j+1}. The decoder is also modeled by a stacked LSTM. Like the encoder, the jth element s_j of the sequence S_i is embedded into a vector by a single fully connected layer f_{demb}, and is processed by a stacked LSTM. Unlike the encoder, however, the embedded vector is concatenated with a sampled latent vector z and a condition vector C_i before inputting into a stacked LSTM. The concatenated vector is fed into the LSTM block f_{dec}, and a sequence S_i is processed by recursively applying the LSTM block f_{dec}. The initial hidden state h_0 is calculated by replacing s_j and h_j with a Start Of Sequence (SOS) and $f_{\text{dinit}}(C_i)$. SOS is converted by f_{dsos} from a vector into which a latent vector z and a condition vector C_i are concatenated together, where the conversion f_{dsos} is implemented by a fully connected layer. The function f_{dinit} is also implemented by a fully connected layer. The output vector of each LSTM block is fed to 5 functions f_{t_u}, f_{t_v}, f_{L_u}, f_{L_e}, and f_{L_v} implemented by a fully connected layer. The 5 vectors output from these 5 functions are respectively converted to probability distributions ξ_{t_u}, ξ_{t_v}, ξ_{L_u}, ξ_{L_e}, and ξ_{L_v} through a softmax function, and these distributions predict one-hot vectors of 5-tuples in s_j. In the learning process, the stacked LSTM and the fully connected layers are trained to predict s_{j+1} by a concatenated vector $\tilde{s}_i = [\xi_{t_u}, \xi_{t_v}, \xi_{L_u}, \xi_{L_e}, \xi_{L_v}]$ (See Section V-D for detailes). Summarizing the above, the process of the decoder part of our model is as follows:

$$\text{SOS} = f_{\text{dsos}}([z^T, C_i^T]^T), \tag{6}$$

$$h_0 = f_{\text{dec}}(f_{\text{dinit}}(C_i), [f_{\text{demb}}(\text{SOS})^T, z^T, C_i^T]^T), \tag{7}$$

$$h_{j+1} = f_{\text{dec}}(h_j, [f_{\text{demb}}(s_j)^T, z^T, C_i^T]^T), \tag{8}$$

$$\xi_{t_u} = \text{Softmax}(f_{t_u}(h_j)), \tag{9}$$

$$\xi_{t_v} = \text{Softmax}(f_{t_v}(h_j)), \tag{10}$$

$$\xi_{L_u} = \text{Softmax}(f_{L_u}(h_j)), \tag{11}$$

$$\xi_{L_e} = \text{Softmax}(f_{L_e}(h_j)), \tag{12}$$

$$\xi_{t_v} = \text{Softmax}(f_{L_v}(h_j)), \tag{13}$$

$$\tilde{s}_j = [\xi_{t_u}^T, \xi_{t_v}^T, \xi_{L_u}^T, \xi_{L_e}^T, \xi_{L_v}^T]^T. \tag{14}$$

D. Training

Using sequence data fed by DFS code and structural feature vectors calculated by a statistical process, GraphTune infers the functions mentioned in Section V-C. In the training process, we input sequence data $S_i \in \mathcal{S}$ and a condition vector $C_i \in \mathcal{C}$ into the proposed model, and obtain a latent vector z and a predicted sequence $\tilde{S}_i = \{\tilde{s}_j | j = 0, 1, \cdots, k\}$.

Following the optimization manner of VAE [22], our model with encoder $q_\phi(z|S_i)$ and decoder $p_\theta(S_i|z)$ considers the two components

$$-\mathbb{D}_{\text{KL}}(q_\phi(z|S_i)||p(z)) \quad \text{and} \tag{15}$$

$$\mathbb{E}_{q_\phi(z|S_i)}[p_\theta(S_i|z)] \tag{16}$$

of the variational lower-bound, where $\mathbb{D}_{\text{KL}}(\cdot)$ and $p(z)$ denote the Kullback-Leibler divergence and the multidimensional standard normal distribution $\mathcal{N}(\mathbf{0}, I)$ with dimension L, respectively. Like the normal VAE [22], the first component Eq. (15) regularizes the latent state distribution to be the standard normal distribution, and can be written as

$$\text{Loss}_{\text{enc}}(\mu, \sigma) = \frac{1}{2} \sum_{l=1}^{L} (1 + \log((\sigma_l)^2) - (\mu_l)^2 - (\sigma_l)^2). \tag{17}$$

The second component Eq. (16) is a reconstruction loss that ensures the predicted sequence is similar to the input sequence from the dataset. For our model, the reconstruction loss is defined for a sequence data S_i and a predicted sequence $\tilde{S}_i$ by

$$\text{Loss}_{\text{dec}}(\tilde{S}_j, S_j) = -\frac{1}{|S|} \sum_{i=0}^{|S|} \sum_c s_j(c) \log(\tilde{s}_j(c)), \tag{18}$$

where $s(c)$ and $\tilde{s}(c)$ represent component $c \in \{t_u, t_v, L_u, L_e, L_v\}$ of s_i and $\tilde{s}_i$, respectively.

By uniting the two losses with the idea of β-VAE [31], the proposed model is optimized by gradient descent on the following loss with weight β.

$$\text{Loss}(\mu, \sigma, \tilde{s}_j, s_j) = \beta \cdot \text{Loss}_{\text{enc}}(\mu, \sigma) + \text{Loss}_{\text{dec}}(\tilde{s}_j, s_j). \tag{19}$$

The loss is backpropagated to the model, and we use the reparameterization trick [22] for backpropagation through the Gaussian latent variable.

The detailed algorithm of the training is shown in Algorithm 1. For a given sequence dataset $\mathcal{S} = \{S_1, S_2, \ldots\}$ and the condition vector set $\mathcal{C} = \{C_1, C_2, \ldots\}$ corresponding to the dataset, Algorithm 1 returns learned encoder functions (f_{eemb}, f_{enc}, f_μ, and f_{σ^2}) and decoder functions (f_{dinit}, f_{demb}, f_{dec}, f_{t_v}, f_{t_v}, f_{L_u}, f_{L_e}, and f_{L_v}). First, the total loss is initialized (Line 2). The algorithm then iterates over all sequences S_i (Lines 3-23). An encoder recursively calculates h_j and a latent vector z is sampled (Lines 4-9). The loss for regularization of

---

Algorithm 1 Training of GraphTune

Input: Graph dataset $\mathcal{S} = \{\boldsymbol{S}_1, \boldsymbol{S}_2, \dots\}$,
    Condition vector set $C = \{\boldsymbol{C}_1, \boldsymbol{C}_2, \dots\}$
Output: Learned functions f_{eemb}, f_{enc}, f_{μ}, f_{σ^2}, f_{dinit}, f_{demb},
    f_{dec}, f_{t_u}, f_{t_v}, f_{L_e}, and f_{L_v}

1: **repeat**
2:    Loss $\leftarrow 0$
3:    **for** i from 1 to $|\mathcal{S}|$ **do**
4:       $\boldsymbol{h}_0 \leftarrow \boldsymbol{0}$
5:       **for** j from 0 to $k-1$ **do**
6:          $\boldsymbol{h}_{j+1} \leftarrow f_{\text{enc}}(\boldsymbol{h}_j, f_{\text{eemb}}([\boldsymbol{s}_j^T, \boldsymbol{C}_i^T]^T))$
7:       **end for**
8:       $\boldsymbol{\mu} \leftarrow f_{\mu}(\boldsymbol{h}_k);\ \boldsymbol{\sigma}^2 \leftarrow f_{\sigma^2}(\boldsymbol{h}_k)$
9:       $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$
10:      Loss $\leftarrow$ Loss $+ \beta \cdot$ Loss$_{\text{enc}}(\boldsymbol{\mu}, \boldsymbol{\sigma})$
11:      $\tilde{\boldsymbol{S}}_i \leftarrow []$
12:      SOS $\leftarrow f_{\text{dsos}}([\boldsymbol{z}^T, \boldsymbol{C}_i^T]^T)$
13:      $\boldsymbol{h}_0 \leftarrow f_{\text{dec}}(f_{\text{dinit}}(\boldsymbol{C}_i), [f_{\text{demb}}(\text{SOS})^T, \boldsymbol{z}^T, \boldsymbol{C}_i^T]^T)$
14:      **for** j from 0 to $k-1$ **do**
15:         **for** $c \in (t_u, t_v, L_u, L_e, L_v)$ **do**
16:            $\boldsymbol{\xi}_c \leftarrow \text{Softmax}(f_c(\boldsymbol{h}_j))$
17:         **end for**
18:         $\tilde{\boldsymbol{s}}_j \leftarrow [\boldsymbol{\xi}_{t_u}^T, \boldsymbol{\xi}_{t_v}^T, \boldsymbol{\xi}_{L_u}^T, \boldsymbol{\xi}_{L_e}^T, \boldsymbol{\xi}_{L_v}^T]^T$
19:         $\tilde{\boldsymbol{S}}_i \leftarrow [\tilde{\boldsymbol{S}}_i, \tilde{\boldsymbol{s}}_j]$
20:         $\boldsymbol{h}_{j+1} \leftarrow f_{\text{dec}}(\boldsymbol{h}_j, [f_{\text{demb}}(\boldsymbol{s}_j)^T, \boldsymbol{z}^T, \boldsymbol{C}_i^T]^T)$
21:      **end for**
22:      Loss $\leftarrow$ Loss $+$ Loss$_{\text{dec}}(\tilde{\boldsymbol{S}}_i, \boldsymbol{S}_i)$
23:    **end for**
24:    Back-propagate Loss and upate weights
25: **until** stopping criteria

---

the latent state distribution is added to the total loss (Line 10). A predicted sequence $\tilde{\boldsymbol{S}}_i$, SOS and $\boldsymbol{h}_0$, are initialized by an empty vector, f_{dsos}, and f_{dinit}, respectively (Line 11-13). $\boldsymbol{h}_j$ is converted to probability distributions of one-hot vectors of 5-tuples through the function f_{t_v}, f_{t_v}, f_{L_u}, f_{L_e}, and f_{L_v} (Lines 15-17). Next, the distribution is vertically concatenated into a single vector $\tilde{\boldsymbol{s}}_j$, and these vectors $\{\tilde{\boldsymbol{s}}_j | j = 1, 2, \dots, |\boldsymbol{S}_i|\}$ are horizontally concatenated into a predicted sequence $\tilde{\boldsymbol{S}}_i$ (Lines 18-19). A decoder also recursively calculates $\boldsymbol{h}_j$ for the prediction of the next 5-tuple (Line 20). The reconstruction loss calculated from the concatenated probability distributions is added to the total loss (Line 22). Lastly, the weights of all functions are updated by back-propagating the total loss (Line 24). The above procedures are iterated until the total loss converges (Line 25).

E. Generation

When we generate graphs with specific structural features, GraphTune recursively generates sequential data in the DFS code format using learned functions. We give a sampled latent vector and a condition vector whose elements are tuned to specific values to the decoder. By giving a condition vector, the decoder recursively generates a sequence of 5-tuples according to the condition vector. Finally, we get a graph with specific

structural features by inverse converting from sequence data to a graph.

The entire procedure for the generation of a graph with a specific condition is summarized in Algorithm 2. The input and output of the algorithm are a condition vector $\boldsymbol{C}$ with specific values and sequence data of a graph with the condition, respectively. Firstly, a generated sequence data $\boldsymbol{S}$ and an iterator variable j are initialized (Lines 1-2). For the generation, a latent vector $\boldsymbol{z}$ is sampled from the standard normal distribution $\mathcal{N}(\boldsymbol{0}, \boldsymbol{I}^2)$ (Line 3). $\boldsymbol{h}_0$ is calculated from the sampled latent vector $\boldsymbol{z}$ and the given condition $\boldsymbol{C}$ (Lines 4-5). To obtain the next 5-tuple in a predicted sequence, the element-wise distributions $\xi_{t_u}, \xi_{t_v}, \xi_{L_u}, \xi_{L_e}$, and ξ_{L_v} of 5-tuple are calculated, and predicted values $\hat{\boldsymbol{s}}(t_u), \hat{\boldsymbol{s}}(t_v), \hat{\boldsymbol{s}}(L_u), \hat{\boldsymbol{s}}(L_e)$, and $\hat{\boldsymbol{s}}(L_v)$ are sampled from these distributions, respectively (Lines 7-10). The predicted values $\hat{\boldsymbol{s}}(t_u), \hat{\boldsymbol{s}}(t_v), \hat{\boldsymbol{s}}(L_u), \hat{\boldsymbol{s}}(L_e)$, and $\hat{\boldsymbol{s}}(L_v)$ of each element of 5-tuple are vertically concatenated into a single vector $\hat{\boldsymbol{s}}_j$, and the concatenated vector $\hat{\boldsymbol{s}}_j$ is horizontally concatenated to the predicted sequence $\boldsymbol{S}$ (Lines 11-12). $\boldsymbol{h}_j$ is recursively calculated for the prediction of the next 5-tuple, and the iterator variable j is updated (Lines 13-14). To stop the generation of the sequence in finite size, the iteration finishes if at least one element of the prediction $\hat{\boldsymbol{s}}_j$ of 5-tuple is EOS (Line 15). A graph with a specific condition $\boldsymbol{C}$ is easily constructed from the sequence $\boldsymbol{S}$ at the end of the algorithm.

Most parts of the algorithm of GraphTune are deterministic, but the latent vector $\boldsymbol{z}$ and the 5-tuple in DFS code are randomly sampled from estimated distributions. By the sampling process, GraphTune is possible to generate a wide variety of graphs under a single condition. Based on the basic concept of VAE, the final output of the encoder of GraphTune is not the latent vector $\boldsymbol{z}$, but the parameters $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ of the normal distribution over the latent space. The latent vector $\boldsymbol{z}$ is sampled from the normal distribution specified by the parameters. Similarly, the decoder output in GraphTune is a distribution from which 5-tuples in DFS code will be sampled.

In the generating process of a graph from a DFS code, we construct the graph by creating nodes and edges based on the timestamps of the nodes in the DFS code that is output by the decoder. Since this research focuses on generating unlabeled graphs, we ignore label information in the DFS code. Although GraphTune attempts to faithfully reproduce the DFS code in the graphs of the dataset, GraphTune can not guarantee full compliance with the rules of DFS code. Therefore, 5-tuples that violate the rule of DFS code may be output. A typical case is that a 5-tuple contained in the DFS code reappears in subsequent DFS code. To solve these problems, we ignore any 5-tuple that conflicts with the 5-tuples that appear in the DFS code before that.

F. Necessity of CVAE Architecture

The most distinctive point of GraphTune is that it allows continuous tuning of a structural feature of generated graphs by giving condition vectors to the decoder and the encoder in VAE, respectively. Although the architecture of Graph-Tune is similar to that of GraphGen, GraphTune differs from

Algorithm 2 Generation of a graph with a specific condition

Input: Condition vector $\boldsymbol{C}$ with specific values
Output: Sequence data $\boldsymbol{S}$ of a graph G with a specific
    condition $\boldsymbol{C}$

  1: $\boldsymbol{S} \leftarrow []$
  2: $j \leftarrow 0$
  3: $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}^2)$
  4: SOS $\leftarrow f_{\text{dsos}}([\boldsymbol{z}^T, \boldsymbol{C}^T]^T)$
  5: $\boldsymbol{h}_0 \leftarrow f_{\text{dec}}(f_{\text{dinit}}(\boldsymbol{C}), [f_{\text{demb}}(\text{SOS})^T, \boldsymbol{z}^T, \boldsymbol{C}^T]^T)$
  6: **repeat**
  7:    **for** $c \in (t_u, t_v, L_u, L_e, L_v)$ **do**
  8:       $\boldsymbol{\xi}_c \leftarrow \text{Softmax}(f_c(\boldsymbol{h}_j))$
  9:       $\hat{s}(c) \sim \boldsymbol{\xi}_c$
 10:    **end for**
 11:    $\hat{\boldsymbol{s}}_j \leftarrow [\hat{s}(t_u)^T, \hat{s}(t_v)^T, \hat{s}(L_u)^T, \hat{s}(L_e)^T, \hat{s}(L_v)^T]^T$
 12:    $\boldsymbol{S} \leftarrow [\boldsymbol{S}, \hat{\boldsymbol{s}}_j]$
 13:    $\boldsymbol{h}_{j+1} \leftarrow f_{\text{dec}}(\boldsymbol{h}_j, [f_{\text{demb}}(\hat{\boldsymbol{s}}_j)^T, \boldsymbol{z}^T, \boldsymbol{C}^T]^T)$
 14:    $j \leftarrow j + 1$
 15: **until** $\exists c : \hat{s}(c) = \text{EOS}$

GraphGen in that it is divided into an encoder and a decoder. Our goal is to tune only the feature that is specified in the condition vector while maintaining the values of the other features. In order to achieve this goal, it is important not only to simply learn graph features, but also to separate the features into the feature specified in the condition vector (conditioned features) and the other features (unconditioned features). Since the decoder of GraphTune is given a condition vector along with a latent vector, it can proceed with learning to reduce the reconstruction loss based on accurate information regarding conditioned features. Therefore, the latent vector does not have to contain information about conditioned features. On the contrary, it is desirable that the latent vector contains only information about unconditioned features. In GraphTune, by explicitly giving the encoder a condition vector as well as the decoder, the encoder can remove information about the conditioned features from the latent vector.

The architecture of GraphTune, which provides condition vectors to both the encoder and the decoder in VAE, is a reasonable architecture for accurately tuning features of generated graphs. An architecture that inputs a condition vector to either the encoder or the decoder does not have significant tunability. The detailed experimental results that support the necessity of the CVAE-based architecture of GraphTune are presented in Section VI-E.

VI. Experiments

We verify that GraphTune can learn structural features from graph data and generate a graph with specific structural features. In this section, we first present performance evaluations of GraphTune on a graph dataset generated with WS model [8], which is representative of statistical graph generation models. In addition, we also present a performance evaluation of GraphTune on a real graph dataset extracted from a who-follows-whom network of Twitter. Through the evaluations, we show that GraphTune yields better performance

than the conventional generative models, namely, GraphGen and CondGen.

A. Baselines

To confirm the basic characteristics of GraphTune in a conditional graph generation task, we compare the performance of GraphTune with two baseline models: GraphGen [17] and CondGen [15].

GraphGen: GraphGen employs a scalable approach to domain-agnostic labeled graph generation and is a representative model that adopts a sequence data-based approach. As we mentioned in the introduction, the sequence data-based approach is one of the most successful approaches in the field of learning-based graph generation. GraphGen was compared with DeepGMG [12] and GraphRNN [11] in [17]. It was reported that GraphGen is superior to these methods in terms of the reproduction accuracy of graph structural features. Although GraphGen is an outstanding model, it unfortunately does not provide conditional generation of graphs. Hence, GraphGen is a baseline in terms of the reproduction accuracy of graph structural features, and it does not provide a baseline regarding the ability of conditional generation. In the evaluations in Section VI, we use the parameters recommended in [17].

CondGen: CondGen employs conditional structure generation through graph variational generative adversarial nets and is one of the few models that achieves a conditional generation for general graphs that is not limited to a specific domain. To the best of our knowledge, CondGen is almost the only model that is oriented towards the reproduction of global-level structural features in human relationship graphs including social networks and citation networks. GraphGen was compared with GraphVAE [13], NetGAN [5], and GraphRNN [11] in [15]. The study [15] reports that CondGen records the best performance in most cases. Since CondGen supports a conditional generation of graphs, CondGen provides a baseline regarding the ability of conditional generation. Unlike GraphTune specifying a value of a feature after a training process, CondGen requires training datasets grouped by labels. Hence, when we specify another condition, CondGen needs to relearn another dataset grouped by the conditions. In the evaluations in Section VI, we use the parameters recommended in the paper [15]. Since the number of nodes and edges are required as input parameters in the generation, the number of nodes and edges of sampled graphs with a specific label from the dataset are used.

B. Parameters and Training Dataset

The parameters of a model in GraphTune are set as follows. For the encoder part of our model, we use 2-layer LSTM blocks for f_{enc}, which has a hidden state vector of dimension 223. The dimension of f_{eemb} is set to 227. The dimensions of vectors $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$, that is, the dimension of the latent vector $\boldsymbol{z}$, are set to 10. Three-layer LSTM blocks for f_{dec} which have a hidden state vector of dimension 250, is adopted for the decoder part. The dimensions of f_{dsos} and f_{demb} are set to 250. We use Adam optimizer to train the neural networks for

10000 epochs with a batch size of 37 and an initial learning rate of 0.001 for training. The weight β for the calculation of loss is set to 3.0.

To investigate the basic performance of GraphTune, we constructed a graph dataset (WS dataset) generated by WS model. As the parameters of WS model, we set average degree K to 3, and randomly change rewiring probability p within the range $[0.1, 0.6]$. As we mentioned in the introduction, since stochastic graph generation models generate graphs with simple algorithms, they focus on only a single-aspect feature. Therefore, in graph datasets that are generated by these models, a part of the features that are not focused on are almost invariable. In the case of WS model, since it is a model that aims to reproduce small worldness, the average shortest path length changes greatly, but the other features are almost invariable. Our WS dataset contains average shortest path lengths ranging from 5.32 to 17.0.

To evaluate the performance of GraphTune on real graph dataset, we sampled data from the Twitter who-follows-whom graph in the Higgs Twitter Dataset [32]. To prepare a human relationship graph dataset with sufficient size for training and evaluation, we sampled 2000 graphs from a single huge graph included in the Higgs Twitter Dataset with 456,626 nodes and 14,855,842 edges. A graph in the dataset for the evaluations is sampled by performing a random walk that starts from a randomly selected node. An initial node of a random walk is selected following a uniform distribution. The edge for the next hop is randomly selected from all edges with equal probability. The random walk ends up after 50 nodes are found, and a graph in the evaluation dataset is an induced subgraph that is composed of the nodes included in the random walk. Note that an edge can be included in the evaluation dataset if both nodes are included in the random walk, even if the edge is not included in the random walk. Although the original Higgs Twitter Dataset is a directed graph, we ignore the directions of all edges in this study since our model is designed for undirected graphs. This dataset with small and uniform size graphs is suitable for evaluating the basic tunability of global-level features without being affected by the difficulty of reproducing heterogeneous or very large graphs. We split the dataset into two parts: the training set and validation set. The size ratio of the training set and validation set are 90% and 10%, respectively.

C. Structural Features

As structural features of graphs, we focus on the following 5 features: average of shortest path length, average degree, modularity [33], clustering coefficient, and a power-law exponent of a degree distribution [34]. The value of modularity is calculated for modules consisting of nodes divided by the Louvain algorithm [35]. We calculate the power-law exponent of the degree distribution by the powerlaw Python package [34]. These global-level structural features are selected from survey papers [36], [37] on the measurement of complex network structures, and they have been widely used as graph features of human relationship graphs [38], [39]. Compared with local structures such as the number of nodes and edges, these global-level structural features are difficult to tune by adding or

removing local structure to or from a graph, such as a node, edge, or hexagon. Needless to say, adding or removing local structures to or from a graph can change the value of global-level structural features. However, if we control a value of a feature to a specific value on a graph, we must understand the structure of the whole graph and consider the effect of the local structure on the value of the feature. For the creation of the dataset, the value of features is rounded to one decimal place.

D. Performance Evaluations

In this section, we show that GraphTune can generate graphs with specific structural features. Performance comparison among three methods (GraphTune, CondGen, and GraphGen) and detailed analysis of generated graphs are provided.

We trained GraphTune, CondGen, and GraphGen using the training set described in Section VI-B, and generated graphs with specific conditions. For GraphGen, a single model is trained with the training set, since GraphGen does not provide conditional generation of graphs. For GraphTune and CondGen, the models were trained individually for each feature that is focused on; that is, we trained 5 different models for each single feature. After the training process, we generated 300 graphs for each model. For each feature, we picked up 3 typical values of a feature from the range of values of the training set as the values of the condition vectors. In the generation process, we give the condition vectors to models of GraphTune. Since CondGen requires training sets grouped by labels, the training sets are divided into 3 groups at the middle of the typical values. Note that we cannot give a condition to GraphGen since it is designed for unconditional generation.

The summary of the results of generation for WS dataset and Twitter dataset are listed in TABLE I and II, respectively. The values of features other than average shortest path length of graphs in WS dataset are almost invariant, we list only results regarding average shortest path length in TABLE I. In TABLE II regarding the results for Twitter dataset, we list the results for all 5 features shown in Section VI-C. The values in the columns of GraphTune, CondGen, GraphGen represents average values of features in graphs generated by each model. Since GraphGen does not provide conditional generation, the same value is listed for different conditions. We can consider that a method has better performance if the average value is closer to the values of the condition in the tunability of a condition. The best performance achieved under each condition for a particular feature is emphasized in bold font.

As shown in TABLE I, GraphTune well reproduces the shift of the average shortest path length with the change in rewiring probability on WS dataset. The value of the average shortest path length of the graphs generated by GraphTune is close to the value specified by the condition vector. The average shortest path length for CondGen cannot be calculated since most of generated graphs are unconnected. Then, we depict "–" for unconnected graphs. For reference, the average shortest path length of the largest component is given in parenthesis for unconnected graphs. In contrast, all graphs generated by

GraphTune were connected graphs. This is an advantage of the sequence data-based approach taken by GraphTune (and GraphGen). Even comparing the values of the average shortest path length of the largest components, it can be confirmed that the tunability of GraphTune is higher than that of CondGen.

According to the results in TABLE II, we can confirm that the graphs generated by GraphTune have high reproduction accuracy and clearly change depending on the conditions. GraphGen generally reproduces real data well, but GraphTune, which adopts sequence-based generation like GraphTune, has similar performance. A similar conclusion can be drawn from the visualized graphs of each model shown in Fig. 5. In the results of average degree and clustering coefficient, information of condition vector works effectively, and GraphTune has better reproduction accuracy than GraphGen. In tunability of feature, GraphTune achieves the best performance in most of the features. While GraphTune can accurately tune the average of shortest path length, the value for CondGen cannot be calculated since all generated graphs are unconnected. Since we explicitly give information on the number of nodes and edges in the graph of the training dataset, CondGen can accurately tune the value of average degree. GraphTune is also quite accurate even though such information is not given. With regard to modularity and clustering coefficients, GraphTune outperforms CondGen in terms of both reproduction accuracy of real data and tunability. For highly complex statistics such as the power-law exponent, the value of the feature is somewhat tunable but the result is a little unstable.

To investigate the detailed performance of GraphTune, we depict the distributions of the values of the global-level structural features. In Fig. 4, we plot pairwise relationships of the values of the features on generated graphs by GraphTune. While maintaining the value of the other features of the generated graphs are in the range of real data, the distributions of values of an average shortest path length on GraphTune results are clearly distinguishable. According to the scatter plots in Fig. 4, we can confirm that the distribution of points in the real graph data and that of the generated graph data almost overlap. As a result, the relationships between any two features are accurately reproduced, and it was achieved that the reproduction of a graph dataset in every single aspect.

E. Ablation Study

We performed an ablation study to demonstrate the validity of the architecture of GraphTune. When giving a condition to the LSTM-based VAE in GraphTune, a condition vector is input to 3 spots: the input sequence of the encoder (Eq. (2)), the input sequence of the decoder (Eq. (8)), and the hidden layer of the decoder (Eq. (7)). To evaluate the impact of the condition vector on the tunability, we prepared 3 versions of GraphTune by removing one of the 3 spots at which a condition vector is input. We trained each model with Twitter dataset and generated 300 graphs. Each model is trained with the same parameters as the experiment regarding the average shortest path length shown in Section VI-D. We calculated the average shortest path length for all generated graphs, and calculated the Root Mean Squared Error (RMSE) with the

value specified in the condition vector as the true value. The results of RMSE for each model are listed in TABLE III. As shown in TABLE III, RMSE of the original GraphTune is the lowest value among all models. According to the results, although 3 spots of input of a condition vector in the GraphTune architecture seem redundant, we can understand that all condition inputs contribute to improving its tunability. The most significant contributor to the tunability is the condition vector that is concatenated to the input sequence of the encoder. This suggests that removing the information related to the specified feature from the latent vector is important for the graph generation. Meanwhile, the condition vector that is concatenated to the input sequence of the decoder also contributes significantly to the tunability.

F. Latent Space Analysis

Together with the above evaluations regarding tunability, we analyzed the structure of the latent state distribution $\mathcal{Z}$ in GraphTune. The reference [40] focuses to enforce disentangled representations of model parameters in a graph generation based on the idea of β-VAE [31]. Although the proposed model in the reference [40] cannot provide conditional generation of graphs, the authours reported that it is possible to construct a generative model that correlates the number of nodes n and edge density p with the elements of the latent vector z in a simple ER model. We performed an experiment with the graph dataset (ER dataset) generated by the same conditions as ER model in reference [40], and verified the structure of the latent space and generation results. All parameters in GraphTune are the same as those evaluated in Section VI-D, except the dimension of the latent space is set to 4 which is the same as the experimental conditions in reference [40]. In the learning process, we used connected graphs in ER dataset, and gave edge density p as a condition vector.

Fig. 6 shows the relationship between each element z_i of the latent vector z and edge density p, and it is confirmed that the latent space of GraphTune is not disentangled according to the figure. This analysis that visualizes the correlation between z and p is adopted in the evaluation in reference [40]. On the other hand, from the pairwise relationship of n and p shown in Fig. 7, we can clearly confirm the correlation between values of condition vectors and edge density p of the generated graphs. In this result, the correlation coefficient between values of the condition vector and p is 0.95, and it is higher than the correlation coefficient of 0.35 between the element of latent vector z_i and p reported in the reference [40]. It should be noted that p depends on a condition, whereas n can be determined independently of a condition. The above results mean that GraphTune achieves to generate graphs depending only on the value of the condition given to the decoder, not on the information of the latent space.

VII. LIMITATIONS AND FUTURE DIRECTIONS

We recognize that there are some limitations of generation by GraphTune, which suggest that GraphTune has potential for future expansion.

TABLE I
THE AVERAGE SHORTEST PATH LENGTH IN GRAPHS GENERATED BY GRAPHTUNE, CONDGEN, AND GRAPHGEN FOR WS DATASET. WE PRESENT THE
SAME VALUE FOR ALL RESULTS OF GRAPHGEN FOR EACH FEATURE SINCE GRAPHGEN DOES NOT PROVIDE CONDITIONAL GENERATION OF GRAPHS.
THE BEST PERFORMANCE BETWEEN GRAPHTUNE AND CONDGEN ACHIEVED UNDER EACH CONDITION IS HIGHLIGHTED IN BOLD FONT.

Global-level structural feature	Condition	GraphTune average	CondGen average	GraphGen average	WS data average (25-percentile / median / 75-percentile)
Average of shortest path length	7.5	**10.8**	– (2.28)	9.51	9.10 (7.67 / 8.74 / 10.2)
	10.0	**14.5**	– (2.27)		
	12.5	**16.8**	– (2.30)		

TABLE II
AVERAGE VALUES OF 5 GLOBAL-LEVEL STRUCTURAL FEATURES IN GRAPHS GENERATED BY GRAPHTUNE, CONDGEN, AND GRAPHGEN FOR TWITTER
DATASET. WE PRESENT THE SAME VALUE FOR ALL RESULTS OF GRAPHGEN FOR EACH FEATURE SINCE GRAPHGEN DOES NOT PROVIDE CONDITIONAL
GENERATION OF GRAPHS. THE BEST PERFORMANCE BETWEEN GRAPHTUNE AND CONDGEN ACHIEVED UNDER EACH CONDITION FOR A PARTICULAR
FEATURE IS HIGHLIGHTED IN BOLD FONT.

Global-level structural feature	Condition	GraphTune average	CondGen average	GraphGen average	Real data average (25-percentile / median / 75-percentile)
Average of shortest path length	3.0	**3.05**	– (2.18)	4.59	4.26 (3.40 / 4.09 / 4.84)
	4.0	**4.02**	– (2.24)		
	5.0	**5.43**	– (2.27)		
Average degree	3.0	3.26	**2.93**	2.96	3.59 (2.96 / 3.44 / 3.92)
	3.5	3.60	**3.48**		
	4.0	**3.90**	4.51		
Modularity	0.40	**0.389**	0.299	0.567	0.550 (0.509 / 0.563 / 0.617)
	0.55	**0.430**	0.325		
	0.70	**0.507**	0.336		
Clustering coefficient	0.1	**0.177**	0.344	0.0846	0.203 (0.152 / 0.196 / 0.251)
	0.2	**0.181**	0.366		
	0.3	**0.217**	0.409		
Power-law exponent of a degree distribution	2.6	**2.91**	4.10	5.48	4.28 (2.91 / 3.48 / 4.23)
	3.0	**2.98**	3.90		
	3.4	**3.50**	3.80		

TABLE III
RSME FOR EACH GRAPHTUNE VERSION. WE VERIFY 3 VERSIONS BY REMOVING ONE OF THE 3 SPOTS AT WHICH A CONDITION VECTOR IS INPUT. THE
SMALLEST VALUES OF RMSE AMONG ALL MODELS ACHIEVED UNDER EACH CONDITION ARE HIGHLIGHTED IN BOLD FONT.

Global-level structural feature	Condition	GraphTune Original RMSE	GraphTune without condition on encoder RMSE	GraphTune without condition on decoder RMSE	GraphTune without condition on hidden layer RMSE
Average of shortest path length	3.0	**0.70**	3.06	2.63	1.12
	4.0	**1.15**	3.22	2.82	1.38
	5.0	**1.92**	4.17	3.47	**1.92**

Generation of Large Graphs: Although the number of nodes on graphs in our dataset is relatively large compared with the datasets that have been evaluated in studies regarding learning-based conditional generations of graphs, it is small in terms of social networks. In the results [17] of unconditional generation with GraphGen, which adopts sequence-based generation like GraphTune, the average number of nodes of graphs generated by GraphGen is at most 54.01 nodes. While GraphTune generates graphs of almost the same size as the graphs that GraphGen generates, it was unfortunately not able to generate graphs with over 100 nodes. More innovation is needed to overcome this limitation. Hierarchical generation [18] is relatively easy to implement but is expected to be effective and is a promising option. Another promising option is a combination of deep learning and traditional statistical graph generation. We consider that the sequence-based generation in GraphTune has a high affinity and extensibility for both approaches.

Pinpoint Specification of Features: The tunability of graph features in GraphTune is not perfect. While a distribution of feature values of graphs generated by GraphTune has distinctly different peaks, these values are somewhat varied. Accuracy improvements of the specification remain as a future issue. In addition, GraphTune can specify at most one feature, and currently, it has not succeeded in specifying multiple features at the same time. In order to achieve the specification of multiple features, it is necessary to understand the independency and/or dependency between each feature. However, the independency and/or dependency of global-level structural features is very complex, and understanding it is a challenging issue. It goes without saying that analytical results based on graph theory are important for this issue. However, observing the features of graphs generated by GraphTune may reinforce the results of graph theory by a data-driven approach.

Generation of Extrapolation: Although GraphTune generates graphs with a specific feature flexibly, the tunable range of a feature is limited to the range of the feature within the graphs
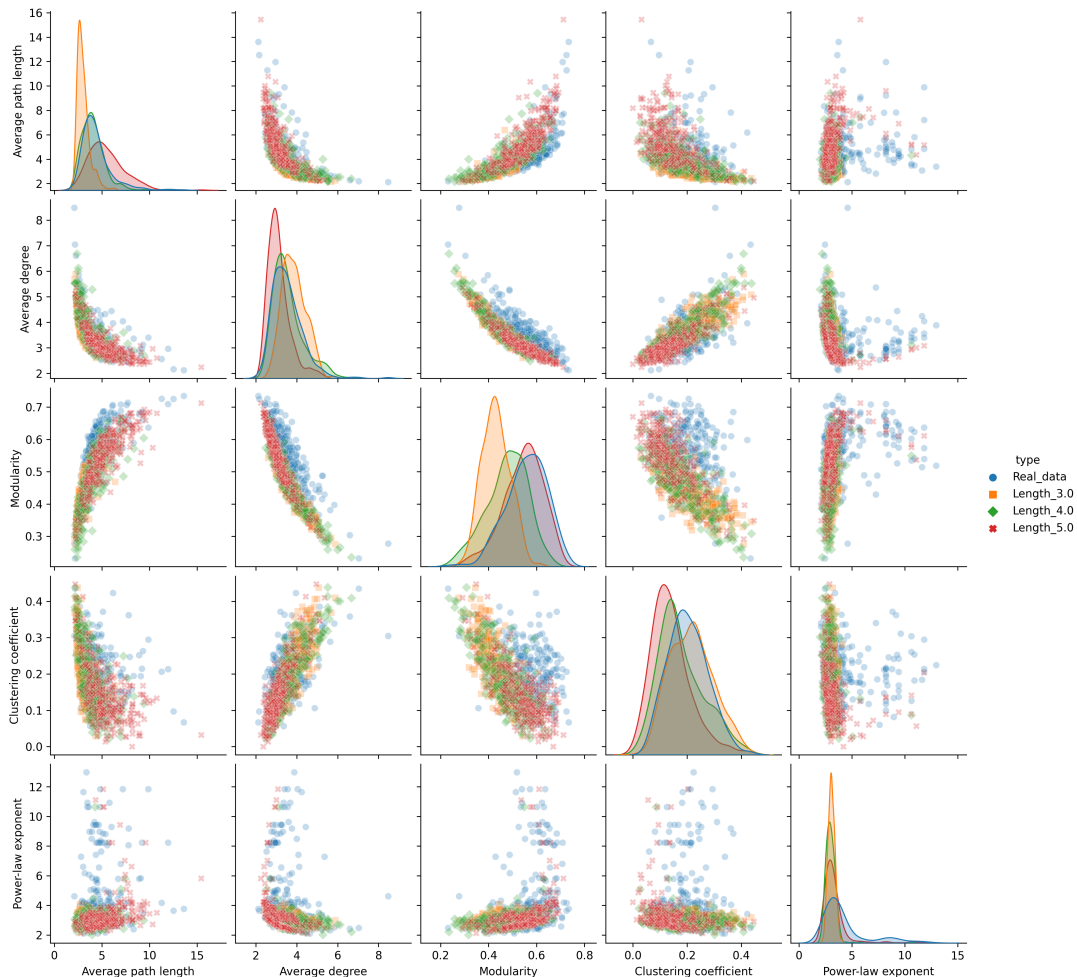
Fig. 4. Pairwise relationships of the values of the features on generated graphs by GraphTune when we give values 3.0, 4.0, and 5.0 of an average shortest path length as conditions. The figure is a grid of multiple plots, and the grid is such that each feature will be shared across the y-axes across a single row and the x-axes across a single column. The diagonal plots are the distributions of the features, and the others are scatter plots of two features. The distributions of values of an average shortest path length are clearly distinguishable.

of the learned dataset. In the tasks of generation or prediction, it is generally hard to extrapolate values that are not included in the range of the dataset. This difficulty is the same for the graph generation task, and the current GraphTune cannot generate extrapolated graphs that are outside the range of graphs included in the training set. However, the specification technique of graph features provided by GraphTune could be a key technology to overcome the difficulty of extrapolation output in the graph generation task. By specifying the value of a feature on the edge of the range of the training dataset, we can generate graphs inside and outside the border. The generated graphs enhance the original training set, and the enhanced training set covers ranges that are not included in the original training set by repeatedly generating graphs on the edges. The generation of extrapolated graphs is one of our future directions.

VIII. CONCLUSION

In this work, we proposed GraphTune, which is a learning-based graph generative model with tunable structural features.

GraphTune is composed of a CVAE with an LSTM-based encoder and decoder. By specifying the value of a particular structural feature as a condition vector that is input into CVAE, we can generate graphs with a specific structural feature. We performed comparative evaluations of GraphTune, CondGen, and GraphGen through a real graph dataset sourced from the who-follows-whom graph on Twitter. The result of the evaluation show that GraphTune makes it possible to tune the value of a global-level structural feature, and that conventional models are unable to tune global-level structural features.

Although GraphTune provides a rich variety of graphs flexibly, it does not solve all problems related to graph modeling. One improvement needed in future works is to provide rich functionality for the specification of structural features. In addition to improving the accuracy of feature values of generated graphs, it is also necessary to be able to specify multiple features at the same time. Allowing the generation of extrapolated graphs that are not included in the training dataset is also an important function. On the other hand, the tunability of GraphTune has the potential to
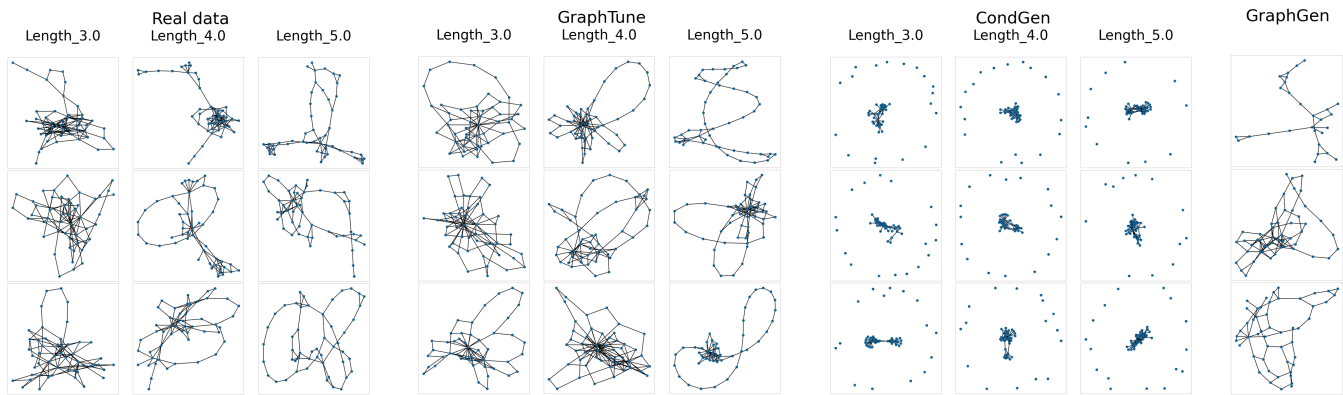
Fig. 5. Real data on the Twitter dataset and generated graphs by each generative model. For all generative models except GraphGen, three samples of generative graphs when we specify the shortest path length to 3, 4, and 5 are depicted. For real data, three samples whose average shortest path lengths are close to 3.0, 4.0, and 5.0 in the dataset are depicted. In Real data and GraphTune, the graphs with "length_3" clearly have a hub node, while the graphs with "Length_5" forms a long loop.
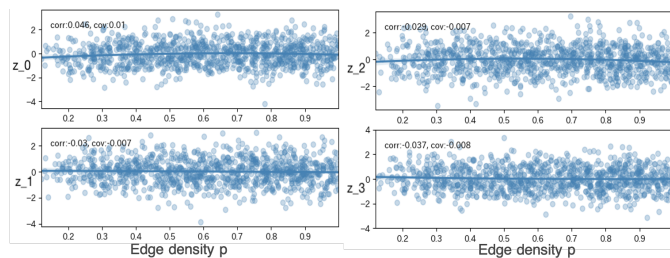

Fig. 6. Relationship between elements z_i of a latent vector z and edge density p of ER model. It is confirmed that the latent space of GraphTune is not disentangled.
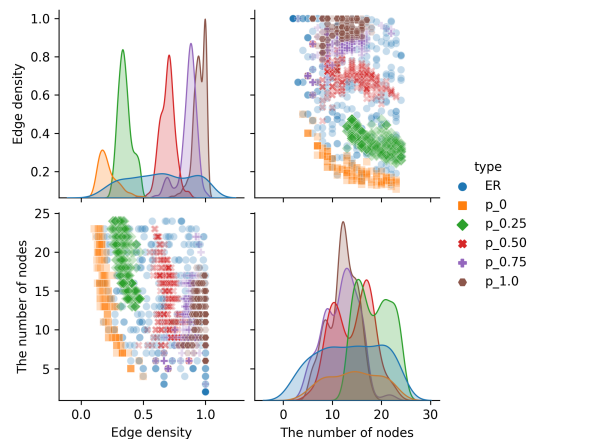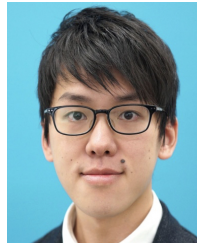

Fig. 7. Pairwise relationships of the values of the features on generated graphs by GraphTune when we give values 0.0, 0.25, 0.5, 0.75, and 1.0 of an edge density p as conditions. The distributions of p are clearly changed depending on a value of a condition vector while the distribution is not significantly dependent on a value of the number n of nodes.

empower traditional graph theory by a data-driven approach. When combined with traditional graph theory, unraveling complex global-level features relationships is a challenging but interesting issue.

REFERENCES

[1] S. Nakazawa, Y. Sato, K. Nakagawa, S. Tsugawa, and K. Watabe, "A Tunable Model for Graph Generation Using LSTM and Conditional VAE," in *Proc. of the 41st IEEE International Conference on Distributed Computing Systems (ICDCS 2021) Poster Track*, 2021.

[2] N. D. Cao and T. Kipf, "MolGAN: An Implicit Generative Model for Small Molecular Graphs," in *Proc. of the 35th International Conference on Machine Learning (ICML 2018) Workshop*, 2018.

[3] Y. Li, L. Zhang, and Z. Liu, "Multi-Objective De Novo Drug Design with Conditional Graph Generative Model," *Journal of Cheminformatics*, vol. 10, no. 33, 2018.

[4] E. Jonas, "Deep Imitation Learning for Molecular Inverse Problems," in *Proc. of the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*, 2019.

[5] 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.

[6] 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.

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

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

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

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

[11] 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.

[12] 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.

[13] 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.

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

[20] 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.

[21] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[22] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," in *Proc. of the 2nd International Conference on Learning Representations (ICLR 2014)*, 2014.

[23] A. Vázquez, "Growing Network with Local Rules: Preferential Attachment, Clustering Hierarchy, and Degree Correlations," *Physical Review E*, vol. 67, no. 5, 2003.

[24] D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-MAT: A Recursive Model for Graph Mining," in *Proc. of the 2004 SIAM International Conference on Data Mining (SDM 2004)*, 2004.

[25] T. G. Kolda, A. Pinar, T. Plantenga, and C. Seshadhri, "A Scalable Generative Graph Model with Community Structure," *SIAM Journal on Scientific Computing*, vol. 36, no. 5, 2014.

[26] J. Liu, Y. Chi, and C. Zhu, "A Dynamic Multiagent Genetic Algorithm for Gene Regulatory Network Reconstruction Based on Fuzzy Cognitive Maps," *IEEE Transactions on Fuzzy Systems*, vol. 24, no. 2, pp. 419–431, 2016.

[27] Y. Wang, W. Che, J. Guo, and T. Liu, "A Neural Transition-Based Approach for Semantic Dependency Graph Parsing," in *Proc. of the 32nd AAAI Conference on Artificial Intelligence (AAAI 2018)*, 2018.

[28] B. Chen, L. Sun, and X. Han, "Sequence-to-Action: End-to-End Semantic Graph Generation for Semantic Parsing," in *Proc. of the 56th Annual Meeting of the Association for Computational Linguistics (ACL 2018)*, 2018, pp. 766–777.

[29] D. Spielman, "Spectral graph theory," in *Combinatorial Scientific Computing*, U. Naumann and O. Schenk, Eds. Taylor & Francis, 2011, ch. 9.

[30] X. Yan and J. Han, "gSpan: graph-based substructure pattern mining," in *Proc. of 2002 IEEE International Conference on Data Mining (ICDM 2002)*, 2002.

[31] I. Higgins, L. Matthey, A. Pal, C. P. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework," in *Proc. of 5th International Conference on Learning Representations (ICLR 2017)*, 2017.

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

[33] M. E. J. Newman and M. Girvan, "Finding and Evaluating Community Structure in Networks," *Physical Review E*, vol. E, no. 69, 2004.

[34] A. Jeff, B. Ed, and P. Dietmar, "powerlaw: A Python Package for Analysis of Heavy-Tailed Distributions," *PLOS ONE*, vol. 9, no. 1, 2014.

[35] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast Unfolding of Communities in Large Networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, 2008.

[36] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D.-U. Hwanga, "Complex Networks: Structure and Dynamics," *Physics Reports*, vol. 424, no. 4-5, pp. 175–308, 2006.

[37] L. da F. Costa, F. A. Rodrigues, G. Travieso, and P. R. V. Boas, "Characterization of Complex Networks: A Survey of Measurements," *Advances in Physics*, vol. 56, no. 1, pp. 167–242, 2007.

[38] H. Kwak, C. Lee, H. Park, and S. Moon, "What is Twitter, a Social Network or a News Media?" in *Proc. of the 19th International Conference on World Wide Web (WWW 2010)*, 2010, pp. 591–600.

[39] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, "On the Evolution of User Interaction in Facebook," in *Proc. of the 2nd ACM Workshop on Online Social Networks (WOSN 2009)*, 2009, pp. 37–42.

[40] N. Stoehr, M. Brockschmidt, J. Stuehmer, and E. Yilmaz, "Disentangling Interpretable Generative Parameters of Random and Real-World Graphs," in *Proc. of the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019) Workshop*, 2019.

Kohei Watabe received his B.E. and M.E. degrees in Engineering from Tokyo Metropolitan University, Tokyo, Japan, in 2009 and 2011, respectively. He also received the Ph.D. degree from Osaka University, Japan, in 2014. He was a JSPS research fellow (DC2) from April 2012 to March 2014. He was an Assistant Professor of the Graduate School of Engineering, Nagaoka University of Technology, from April 2014 to October 2019. He has been an Associate Professor of the Graduate School of Engineering, Nagaoka University of Technology, since November 2019. He is a member of the IEEE and the IEICE.

Shohei Nakazawa received his B.E. and M.E. degrees in Engineering from Nagaoka University of Technology, Niigata, Japan, in 2019 and 2021, respectively.

Yoshiki Sato received his B.E. degree in Engineering from Nagaoka University of Technology, Niigata, Japan, in 2021. He is currently a student at Graduate School of Engineering, Nagaoka University of Technology.

Sho Tsugawa received the M.E. and Ph.D. degrees from Osaka University, Japan, in 2009 and 2012, respectively. He is currently an Assistant Professor with the Faculty of Engineering, Information and Systems, University of Tsukuba, Japan. His research interests include network science, social network analysis, and computational social science. He is a member of IEEE, ACM, IEICE, and IPSJ.

Kenji Nakagawa received the B.S., M.S., and D.S. degrees from Tokyo Institute of Technology, Tokyo, Japan, 1980, 1982, and 1986, respectively. In 1985, he joined Nippon Telegraph and Telephone Corporation (NTT). Since 1992, he has been an Associate Professor with Nagaoka University of Technology, where he has been a Professor since 2012. In 2022, he retired from the Nagaoka University of technology and now he is a Professor Emeritus. His research interests include information theory, performance evaluation of networks, and queueing theory.