PLOS ONE

RESEARCH ARTICLE

SCGG: A deep structure-conditioned graph generative model

Faezeh Faez, Negin Hashemi Dijujin, Mahdieh Soleymani Baghshah*, Hamid R. Rabiee*

Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

* Rabiee@sharif.edu (HRR); Soleymani@sharif.edu (MSB)

Citation: Faez F, Hashemi Dijujin N, Soleymani Baghshah M, Rabiee HR (2022) SCGG: A deep structure-conditioned graph generative model. PLoS ONE 17(11): e0277887. https://doi.org/10.1371/journal.pone.0277887

Editor: Sathishkumar V E, Hanyang University, KOREA, REPUBLIC OF

Received: September 14, 2022

Accepted: November 6, 2022

Published: November 21, 2022

Data Availability Statement: All codes and data used in this manuscript are available at https://gitlab.com/FaezehFaez/scgg.

Funding: Hamid R. Rabiee was partially supported by Iran National Science Foundation (INSF), Grant No. 96006077. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing interests: The authors have declared that no competing interests exist.

Abstract

Deep learning-based graph generation approaches have remarkable capacities for graph data modeling, allowing them to solve a wide range of real-world problems. Making these methods able to consider different conditions during the generation procedure even increases their effectiveness by empowering them to generate new graph samples that meet the desired criteria. This paper presents a conditional deep graph generation method called SCGG that considers a particular type of structural conditions. Specifically, our proposed SCGG model takes an initial subgraph and autoregressively generates new nodes and their corresponding edges on top of the given conditioning substructure. The architecture of SCGG consists of a graph representation learning network and an autoregressive generative model, which is trained end-to-end. More precisely, the graph representation learning network is designed to compute continuous representations for each node in a graph, which are not only affected by the features of adjacent nodes, but also by the ones of farther nodes. This network is primarily responsible for providing the generation procedure with the structural condition, while the autoregressive generative model mainly maintains the generation history. Using this model, we can address graph completion, a rampant and inherently difficult problem of recovering missing nodes and their associated edges of partially observed graphs. The computational complexity of the SCGG method is shown to be linear in the number of graph nodes. Experimental results on both synthetic and real-world datasets demonstrate the superiority of our method compared with state-of-the-art baselines.

1 Introduction

With the ever-increasing growth of data collection and production technologies, large amounts of data are readily accessible. In many cases, some relationship exists between data entities, which, if taken into consideration, can lead to more precise data analyses. Such relationships are mostly represented by graph data structures, and that is why graph-related research has become a widely discussed topic in many areas, including chemistry [1], medical applications [2], social network studies [3], and knowledge graph-related research [4]. Most recent studies are dedicated to graph representation learning [5–9], aiming to obtain suitable

representations of nodes, edges, or the entire graph in continuous space to be further utilized by downstream tasks.

Graph generation is another important branch of graph-related research, which often benefits from the results of graph representation learning studies. This research field has a history of several decades. It has recently been revived by receiving renewed attention from scholars, mainly due to the advances in machine learning, and in particular deep learning techniques. Graph generation aims to provide models that can generate new graph samples from the desired data distributions. Thus, similar to generative methods in other data domains such as image [10], text [11], and speech [12], graph generative approaches can bring substantial capacity for graph data modeling to address various real-world problems such as drug design [13], understanding and modeling the interactions in social networks [14], and human diseases diagnosis [15].

One of the desired and essential properties of generative methods is their ability to carry out the generation procedure in a controlled manner so that the produced samples comply with predetermined conditions by having the required characteristics. In this regard, numerous studies have been conducted to develop conditional generative models in different data domains, such as image [16] and text [17]. Initial steps [18–22] have also been taken to make graph generators conditional. However, compared to the work performed in other data domains and the needs and capacities of this field, much remains to be done.

In addition to what we have discussed so far, there is a common problem manifesting itself when working with different types of data. Specifically, in many cases, the data is not completely available, which can be caused due to various reasons such as limitations of data collection tools, issues related to privacy, or inadequacy of storage space. This can significantly degrade the performance of data analysis methods. Therefore, it is often crucial to recover the missing part of the data before processing it; hence, various methods have been proposed in different data domains to address this challenge. Regarding the graph data, many methods have also been developed for years [23, 24] to predict missing links between graph nodes, and researchers are still seriously pursuing a solution for it [25]. However, an intrinsically more complicated challenge arises when the graph nodes are missing. We will refer to this problem as graph completion, which, unlike the widely investigated problem of link prediction, has been much less addressed despite its importance and pervasiveness.

To address the issues mentioned above, we propose **Structure-Conditioned Graph Generator (SCGG)**, an end-to-end deep learning-based conditional graph generative approach. The SCGG model takes an initial subgraph as the structural condition. It then autoregressively performs the graph generation procedure by adding new nodes and predicting the inter-links between the new nodes and those in the conditioning subgraph, as well as the intra-links between the new nodes. In this way, our generative model ensures the existence of desired subgraphs in the final generated graphs, which can have several applications in both molecular and non-molecular domains. Specifically, for designing molecular graphs, the existence of desired chemical substructures can bring certain chemical properties to the final molecules. Moreover, regarding the non-molecular graphs, the SCGG model can be best utilized to solve the graph completion problem in which some graph nodes and their corresponding edges are totally missing. Our study focuses on the latter application, but the proposed SCGG model can be easily extended to be used in molecular applications as well. In this regard, a partially observed graph is given to the model as a structural condition. Then the generated nodes by the model and their associated edges will be treated as the recovered missing nodes and the edges connecting them to each other, as well as to the partially observed graph nodes.

In summary, we present the following contributions in this work:

- We introduce SCGG, a conditional graph generation approach, which autoregressively generates graphs based on a given structural condition.
- The architecture of our SCGG model consists of a graph representation learning network and a recurrent neural network (RNN), where the former is mainly used to take into account the structural condition, and the latter captures the generation history.
- We use our proposed SCGG model to address the graph completion problem to benefit from the power and potential of a deep generative model for solving an inherently difficult and complex issue, which as a result, has been relatively less investigated so far. To the extent of our knowledge, this is the first time that a completely deep learning-based model is designed in such a way that it can specifically tackle this problem.
- We conduct extensive experiments on both synthetic and real-world datasets to compare the performance of our proposed model against the baselines. The experimental results indicate that the SCGG model outperforms the state-of-the-art baselines in terms of the distance between the generated graphs and the ground-truth ones.

The rest of the paper is organized as follows. In Section 2, we review the previous work related to our research. In Section 3, we introduce the notations used in the paper and define the problem. In Section 4, we explain our proposed SCGG model in detail. Experimental details and results are discussed in Section 5. Finally, in Section 6 we conclude the paper.

2 Related work

In line with what we discussed earlier, our proposed SCGG model is a structure-based conditional graph generation approach, and one of its main applications is graph completion. Therefore, we review the literature in two related areas in the following.

2.1 Graph generation

Graph generation is a field of research seeking to generate new graph structures with certain characteristics, which dates back to several decades ago, and is still a hot topic for research. In contrast to the early methods [26–29], which relied on manually-designed procedures to construct graphs with predetermined statistical properties, the more recent ones are data-driven, utilizing the available graph samples in datasets to train models that can more effectively generate new graphs. The latter approaches typically employ different deep learning techniques and generation strategies, and accordingly, they can be classified into several categories [30]. The autoregressive approaches, which adopt step-by-step strategies for generating graphs, are the most relevant methods to our research. DeepGMG [31] is an example of them proposing a repetitive decision-making process to generate graphs gradually. GraphRNN [32] is among the well-known and influential approaches, which first maps each graph into a sequence of nodes and then processes one node per time step using RNNs to model the distribution of the resulting sequences. The method has inspired several subsequent approaches like MolecularRNN [33], which extends GraphRNN to generate molecular graphs with specific chemical features. Bacciu et al. [34], GraphGen [35], and GHRNN [36], on the other hand, convert graphs to sequences of edges instead of nodes, and then go through distribution modeling with RNNs. Besides, some other autoregressive methods utilize the attention mechanism to empower their generative models. Regarding this, GRAN [37] proposes to add a block of new nodes in each step, and to compute the representations of the graph nodes, it employs an attentive message passing mechanism.

In addition to the methods mentioned above that are more related to our proposed approach, there are other categories of modern graph generation approaches, the most noteworthy of which are autoencoder-based methods [18, 38–42], RL-based approaches [43–45], GAN-based generating strategies [15, 19, 46], and flow-based models [47, 48].

A key point to notice is that regardless of what category these methods fall into and what techniques they employ to solve the problem, an important capability of them is to consider desired conditions during generation so that the resulting graphs meet the expected characteristics. Hence, the problem of conditional graph generation arises. In this regard, GraphVAE [18] conditions both the encoder and the decoder of its VAE on a label vector for the molecular graph generation. CONDGEN [19] adopts a similar approach (i.e., concatenating a condition vector to the VAE latent variable) to incline the model towards generating graphs with desired characteristics. Lim et al. [20], and HierVAE [21] guarantee the existence of intended chemical substructures in the output molecular graphs. CCGG [22] makes the GRAN [37] model class-conditional, allowing it to generate graphs of desired classes. However, despite the efforts that have gone into conditional graph generation, there is still a vital need to develop more and more approaches that can capture various types of conditions. In this regard, the SCGG model is a generative method designed to handle special conditions, which are of structural type.

2.2 Graph completion

In many cases, a part of a graph structure is unavailable for various reasons. Hence, it is necessary to reconstruct the missing information before further processing. Most of the methods developed for this purpose try to perform link prediction [49–51], although a more complicated problem arises when the graph nodes are missing. Therefore, due to the complexity of addressing this problem, which we refer to as graph completion, few methods have been presented to solve it. Regarding this, KronEM [52] utilizes a combination of the Expectation-Maximization framework and the Kronecker graphs model to infer the missing nodes and their corresponding edges. SAMI [53] adopts a clustering approach for solving the missing node problem by heavily relying on the existence of missing node indicators, which are often unattainable in real scenarios. Masrour et al. [54], and JCSL [55] utilize side information about the graph nodes to perform network completion; however, this information may not be accessible in all cases. More recently, DeepNC [56] was introduced, which first learns the likelihood of the data by training the GraphRNN [32] model. It then uncovers the missing parts of a graph by proposing a greedy optimization algorithm, aiming to maximize the obtained likelihood. Although DeepNC is an innovative approach that has obtained satisfactory results, it is not learning-based, so it cannot directly learn from the data for the specific task of graph completion. However, our proposed method trains an end-to-end model to address this problem. Furthermore, unlike some graph completion methods mentioned above, the SCGG does not depend on the existence of side information, which may not be reachable in many situations.

3 Notations and problem definition

In this section, we define the notations used in the paper and present the problem definition. For convenience, we summarize the notations in Table 1.

We denote an initial graph as $G_0 = (V_0, E_0)$, where V_0 and E_0 are the node and the edge sets, respectively, and $|V_0| = n$. Under an ordering π_n of these n nodes, we represent the i-th node's

Table 1. Notations in this paper.

Notation	Description
G_0	An initial graph.
V_0	The node set of G_0.
E_0	The edge set of G_0.
n	The number of nodes in G_0, n = $\lvert V_0 \rvert$.
π_n	An ordering of G_0 nodes.
$N_i^{\pi_n}$	The sequence representing how the i-th node of G_0 under the ordering π_n connects to G_0's nodes.
G	The graph that contains an initial graph G_0 as a subgraph.
V	The node set of G.
E	The edge set of G.
G	The random variable associated with graph structures.
$\tilde{V}$	The set of new nodes added to G_0 to form the graph G.
$\tilde{E}$	The set of edges connecting the new nodes to each other, as well as to those nodes in G_0.
m	The number of new nodes, m = $\lvert \tilde{V} \rvert$.
π_m	An ordering of the new nodes $\tilde{V}$.
$\tilde{N}_i^{\pi_n}$	The sequence representing the links connecting the i-th node of G_0 under the ordering π_n to the new nodes ordered by π_m.
$\tilde{M}_j^{\pi_m}$	The sequence representing the links between the j-th new node and each new node under the ordering π_m.
$S_0^{\pi_n}$	The notational abbreviation for $\{N_1^{\pi_n}, \cdots, N_n^{\pi_n}\}$.
$\tilde{S}^{\pi_n,\pi_m}$	The notational abbreviation for $\{\tilde{N}_1^{\pi_n}, \cdots, \tilde{N}_n^{\pi_n}, \tilde{M}_1^{\pi_m}, \cdots, \tilde{M}_m^{\pi_m}\}$
G'	The graph induced from G by removing the intra-connections between the set of new nodes

https://doi.org/10.1371/journal.pone.0277887.t001

links by the following sequence:

$$N_i^{\pi_n} = (x_k)_{k=1}^{n}, \;\; x_k \in \{0, 1\} \tag{1}$$

where x_k takes value of 1 if the i-th node is connected to the k-th node and 0 otherwise.

Considering G_0 as the structural condition, the objective of our research is to learn to sample from the conditional probability distribution $P(G|G_0)$ in order to generate graph $G = (V, E)$, which includes G_0 as a subgraph, i.e., $V_0 \subset V$ and $E_0 \subset E$. This can be done by first adding the node set $\tilde{V}$, with $|\tilde{V}| = m$ and $\tilde{V} = V - V_0$. Then, to connect new nodes, the edge set $\tilde{E}$ will be generated, where $\tilde{E} = E - E_0$. More specifically, $\tilde{E}$ consists of: 1. the inter-connections between new nodes and those in G_0, 2. the intra-connections between the new nodes themselves. To represent the inter-connections between new nodes and the i-th node of G_0 under the ordering π_n, we use the below sequence:

$$\tilde{N}_i^{\pi_n} = (x_l)_{l=1}^{m}, \;\; x_l \in \{0, 1\} \tag{2}$$

where we consider a node ordering π_m of the m new nodes, and x_l is 1 if the i-th node of G_0 has a link to the l-th new node and 0 otherwise. Moreover, regarding the intra-connections, we denote the j-th new node's connections to the nodes in $\tilde{V}$ by the following sequence:

$$\tilde{M}_j^{\pi_m} = (x_p)_{p=1}^{m}, \;\; x_p \in \{0, 1\} \tag{3}$$

where similarly to the previous formulas, x_p takes the value of 1 if there is a link connecting the j-th and the p-th new nodes (under the ordering π_m) and 0 otherwise.

4 SCGG: Structure-conditioned graph generator

We approach a specific type of structure-conditioned graph generation that takes an initial substructure and starts to generate new nodes and their associated edges on top of the given conditioning substructure. To this end, we propose the SCGG model, whose architecture is composed of a graph representation learning network and an autoregressive generative model, which is trained end-to-end. In this section, we present the details of the SCGG model. We first elucidate the problem formulation and the model architecture in this regard. Next, we describe the procedure employed to prepare the data for model training. Then, we discuss the training and inference phases and elaborate on the implementation details.

4.1 Formulation

As mentioned in the Section 3, in this work we intend to learn to sample from the distribution $P(G|G_0)$ to conditionally generate the graph G given an arbitrary initial graph G_0. To do so, our SCGG model first estimates this conditional probability distribution and then samples from the resulting estimated distribution. As it is not easy to work directly in the graph space, we reformulate the problem to deal with the following distribution:

$$P(\tilde{S}^{\pi_n,\pi_m}|S_0^{\pi_n}) = P(\tilde{N}_1^{\pi_n},\cdots,\tilde{N}_n^{\pi_n},\tilde{M}_1^{\pi_m},\cdots,\tilde{M}_m^{\pi_m}|N_1^{\pi_n},\cdots,N_n^{\pi_n}) \tag{4}$$

where $S_0^{\pi_n}$ and $\tilde{S}^{\pi_n,\pi_m}$ are the notational abbreviations for $\{N_1^{\pi_n},\cdots,N_n^{\pi_n}\}$ and $\{\tilde{N}_1^{\pi_n},\cdots,\tilde{N}_n^{\pi_n},\tilde{M}_1^{\pi_m},\cdots,\tilde{M}_m^{\pi_m}\}$, respectively, and the new problem formulation relates to the original one through the below equation:

$$P(G|G_0) = \sum_{\pi_n,\pi_m} P(\tilde{S}^{\pi_n,\pi_m}|S_0^{\pi_n}) \tag{5}$$

To further decompose the probability in Eq 4, we follow the chain rule and therefore this conditional probability can be rewritten as follows:

$$\begin{aligned}
P(\tilde{S}^{\pi_n,\pi_m}|S_0^{\pi_n}) = \\
&P(\tilde{N}_1^{\pi_n}|N_1^{\pi_n},\cdots,N_n^{\pi_n})\times \\
&P(\tilde{N}_2^{\pi_n}|N_1^{\pi_n},\cdots,N_n^{\pi_n},\tilde{N}_1^{\pi_n})\times \\
&\vdots \\
&P(\tilde{N}_n^{\pi_n}|N_1^{\pi_n},\cdots,N_n^{\pi_n},\tilde{N}_1^{\pi_n},\cdots,\tilde{N}_{n-1}^{\pi_n})\times \\
&P(\tilde{M}_1^{\pi_m}|N_1^{\pi_n},\cdots,N_n^{\pi_n},\tilde{N}_1^{\pi_n},\cdots,\tilde{N}_n^{\pi_n})\times \\
&P(\tilde{M}_2^{\pi_m}|N_1^{\pi_n},\cdots,N_n^{\pi_n},\tilde{N}_1^{\pi_n},\cdots,\tilde{N}_n^{\pi_n},\tilde{M}_1^{\pi_m})\times \\
&\vdots \\
&P(\tilde{M}_m^{\pi_m}|N_1^{\pi_n},\cdots,N_n^{\pi_n},\tilde{N}_1^{\pi_n},\cdots.,\tilde{N}_n^{\pi_n},\tilde{M}_1^{\pi_m},\cdots,\tilde{M}_{m-1}^{\pi_m})
\end{aligned} \tag{6}$$

Our proposed SCGG method trains a novel network architecture in an end-to-end manner to model the complex distribution in Eq 6.

4.2 Model architecture

The model architecture of SCGG consists of two main components, namely, a graph representation learning network and an autoregressive generative model (i.e., an RNN). In the following, we explain these components in detail and discuss the role each plays in the task of structure-conditioned graph generation.

4.2.1 Graph feature learning network. The SCGG method needs appropriate representations of graph nodes beforehand to perform distribution modeling. Therefore, it utilizes a graph representation learning network that employs both a graph convolutional network (GCN) and a Transformer network to learn meaningful node features. Below, we give a brief background of GCNs and Transformers. Furthermore, we elaborate on how each of them contributes to obtaining the final nodes' features in our model.

- **Graph Convolutional Network (GCN)**
  It is often difficult to work directly in the complex and discrete graph space. Therefore, in many cases, obtaining continuous representations of nodes, edges, or the whole graph is necessary before any upcoming tasks. Employing Graph Convolutional Networks addresses this problem. The main idea of GCNs originates from the fact that a node's representation can be obtained by taking into account the features of its own and its neighbors. This is because the neighbors in a graph (i.e., directly or indirectly connected nodes) usually share common characteristics and information.
  Formally, the layer-wise propagation rule of GCNs can be generally formulated as below:

$$X^{l+1} = \phi(AX^lW^l) \tag{7}$$

  where $X^l \in \mathbb{R}^{N\times D_l}$ is the nodes' feature matrix at the l-th GCN layer, N is the number of graph nodes, D_l is the number of features obtained for a node by the previous GCN layer, and X^0 is set to be the initial feature matrix given as input to the GCN; $A \in \mathbb{R}^{N\times N}$ is the adjacency matrix [57, 58] or a variant of it [59, 60]; $W^l \in \mathbb{R}^{D_l\times D_{l+1}}$ is the learnable parameter matrix of the l-th GCN layer, which maps D_l feature channels to D_{l+1} channels; ϕ is a non-linear activation function; $X^{l+1} \in \mathbb{R}^{N\times D_{l+1}}$ is the output feature matrix produced by the l-th GCN layer. Considering this background, our proposed Graph Feature Learning Network first applies L layers of GCN to the input graph. This way, a continuous representation is computed for each graph node based on its neighbors' information.

- **Transformer network**
  In this work, we intend to autoregressively model the distribution in Eq 4, which is conditioned on $\{N_1^{\pi_n}, \cdots, N_n^{\pi_n}\}$. We do so by feeding the representations of graph nodes one at a time into the RNN. Thus, to perform conditional distribution modeling in this way, it is necessary to learn rich node representations so that all the graph nodes can make their own contribution to compute each node's embedding. In other words, we need the representation of a node not only to contain the information of its close neighbors, but also to include the information of relatively distant nodes that share similar characteristics. However, an L-layer GCN only considers information in L-hop neighborhoods to obtain node representations, even if there are some dependencies between farther nodes. Therefore, our proposed Graph Feature Learning Network utilizes a Transformer encoder, which has shown promising results in contextualized representation learning. The following gives a quick overview of its architecture and workflow.
  According to [61], the Transformer encoder layer consists of a multi-head attention block and a feedforward network, each followed by a residual addition and a layer normalization. A multi-head attention block consists of multiple attention heads, each working in a separate

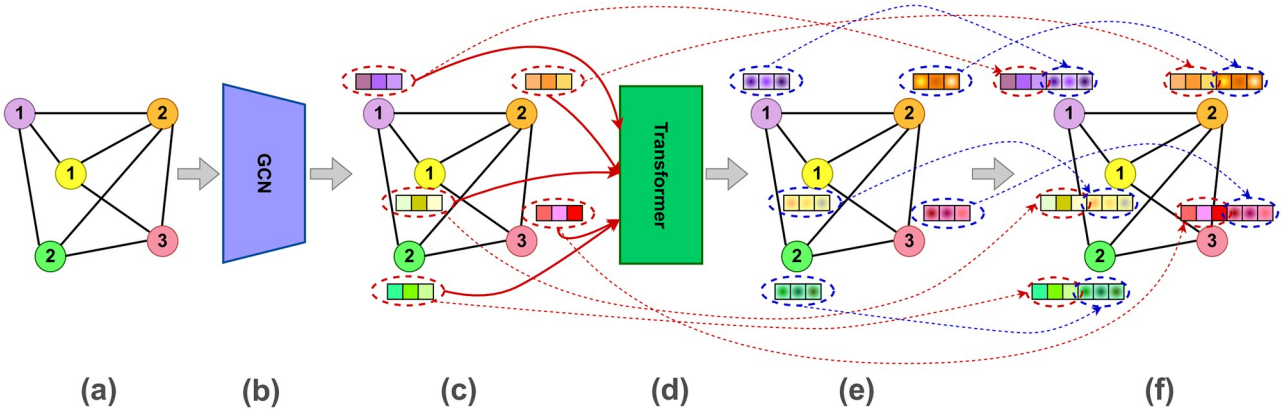

Fig 1. An illustration of the graph feature learning network and its workflow. (a) An input graph. (b) The Graph Convolutional Network. (c) Continuous representations learned for graph nodes by the GCN. (d) The Transformer network takes the node embeddings computed by the GCN as input and outputs new contextualized features of graph nodes. (e) The node features learned by the Transformer network (shown using small squares colored with radial gradients). (f) The final representations of graph nodes acquired by concatenating the embeddings computed by the GCN and the Transformer network. Here, dashed arrows are drawn to easily track what sub-features a final node feature consists of.

https://doi.org/10.1371/journal.pone.0277887.g001

subspace to compute new contextualized representations corresponding to different aspects of dependencies between data entities. To be more precise, each attention head takes as input $X^L \in \mathbb{R}^{N\times D_L}$ (in our case it is the feature matrix computed for graph's nodes by applying L layers of GCN) and projects it into three matrices $Q = X^L W_q \in \mathbb{R}^{N\times D_k}$, $K = X^L W_k \in \mathbb{R}^{N\times D_k}$, and $V = X^L W_v \in \mathbb{R}^{N\times D_v}$ (i.e., query, key, and value, respectively), where W_q, $W_k \in \mathbb{R}^{D_L\times D_k}$ and $W_v \in \mathbb{R}^{D_L\times D_v}$ are learnable matrices. Then, the attention scores for each query are computed over the value matrix V rows by performing an inner product of that query and all the key matrix K rows. By doing so, a new contextualized representation is calculated for each query as a weighted summation of the value matrix rows.

Considering these remarks regarding the GCN and the Transformer, the final nodes' representations are obtained via concatenating the features computed by each of the two networks. Fig 1 shows an overview of the proposed Graph Feature Learning Network.

4.2.2 Autoregressive generative model. As mentioned earlier, we want to model the conditional distribution in Eq 4. To do so, we decompose it as the product of $n + m$ conditional distributions in Eq 6, and then go through modeling them. Each condition in Eq 6 can be divided into two parts: (a) $\{N_1^{\pi_n}, \cdots, N_n^{\pi_n}\}$ that is the initial structural condition regarding to G_0 and (b) The remaining part of the condition that is derived by applying the chain rule, which relates to the generation history. The former is primarily captured by our Graph Feature Learning Network, and the latter is handled using an autoregressive generative model, namely an RNN. More specifically, the embeddings obtained by the Graph Feature Learning Network are fed into the RNN one at a time, and the RNN proceeds. This way, the RNN keeps the generation history such that at each step, the corresponding hidden state maintains the information of the graph generated until that time.

4.3 Data preparation

Making the data suitable as an input to our SCGG model is a prerequisite for training. Therefore, we perform a data preparation procedure before feeding it to the model. This procedure includes determining the set of new nodes $\tilde{V}$, identifying the resulting initial graph G_0, and applying orderings on these two sets of nodes. An example of the data preparation procedure

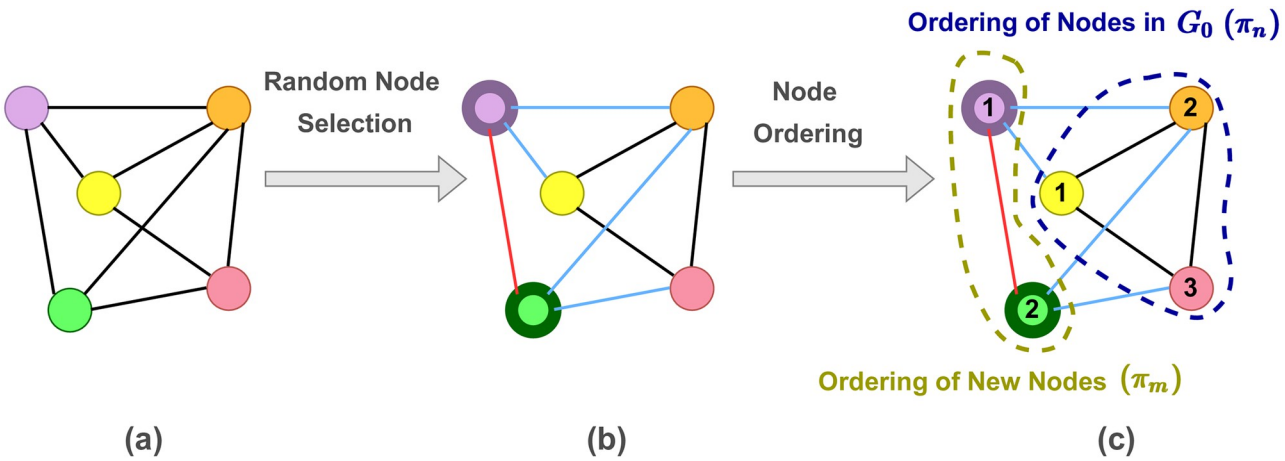

Fig 2. An illustration of the procedure of preparing the training data. (a) An input graph. (b) A number of m nodes are selected randomly to be further treated as the new nodes. In this picture, $m = 2$, and the selected nodes (i.e., the green and the purple ones) are shown with thick borders. Furthermore, the inter-connections between new nodes and those in G_0 are depicted by blue lines, and the only intra-connection between the new nodes is shown using a red line. (c) An ordering π_n is applied to the nodes in G_0. Moreover, another node ordering, denoted by π_m, is applied to the new nodes.

https://doi.org/10.1371/journal.pone.0277887.g002

before model training is illustrated in Fig 2. First, m nodes are randomly selected from the main graph G to form the set of new nodes. Therefore, the n unselected nodes and those edges connecting them are further treated as the initial graph G_0. The reason behind this random node selection is that each subset of n nodes (i.e., the unselected ones) from the original graph has the chance to contribute to the model training as an initial graph. Thus, the model gains the ability to perform structure-conditioned graph generation given an arbitrary graph G_0 at test time. Afterwards, orderings are applied to the nodes such that the initial graph nodes are ordered by π_n, and the new nodes follow the order specified by π_m.

4.4 Training

To train the SCGG model, we first give it two versions of each graph G. The first version corresponds to the initial graph G_0. The second version, which we denote by G', is obtained by removing the intra-connections between pairs of nodes belonging to $\tilde{V}$. The Graph Feature Learning Network takes these two graphs as inputs and separately calculates nodes' representations for each of them, as formulated below:

$$R = [r_1, r_2, \cdots, r_n] = f_{emb}(G_0) \tag{8}$$

$$R' = [r'_1, r'_2, \cdots, r'_n, r'_{n+1}, \cdots, r'_{n+m}] = f_{emb}(G') \tag{9}$$

Next, a subset of the computed representations are fed into the RNN one by one in the order specified by π_n and π_m. More precisely, the RNN first takes the representations of G_0's nodes computed based on the first version of the graph. Then, it receives as input the representations of the new nodes obtained by feeding the G' into the Graph Feature Learning Network. To put it another way, the final representations to be fed into the RNN are as follows:

$$R'' = [r''_i]_{i=1}^{n+m} = [r_1, r_2, \cdots, r_n, r'_{n+1}, \cdots, r'_{n+m}] \tag{10}$$

The reason for this is that at test time, we only have access to an initial graph G_0 knowing nothing about how the set of new nodes are connected to each other as well as to the rest of the graph, but as the RNN proceeds, it predicts the inter-connections between the new nodes and

the nodes of G_0. Thus, when the RNN finishes processing the last node of G_0, all inter-connections have been predicted, and G' can be constructed on top of G_0. At this point, it is time to complete the graph structure by predicting the intra-links between the new nodes. This requires that we have a proper representation for each new node, which can be obtained based on the most complete available version of the graph structure, i.e., the G'.

Moreover, each cell of the RNN takes as its second input the ground truth labels of the previous cell. Therefore, the input for the i-th RNN cell is obtained as follows:

$$x_i = Concat(r''_i, s_{i-1}) \tag{11}$$

where r''_i is the representation of the i-th node and $s_{i-1} \in \mathbb{R}^m$ is the vector of ground truth labels determining whether the $i-1$-th node has links to each of the new nodes or not. Next, by considering both the current input x_i and the previous hidden state h_{i-1}, the RNN outputs probabilities regarding the link existence between the current node and each new node. This is done using two functions f_{RNN} and f_{out} according to the following formulations:

$$h_i = f_{RNN}(x_i, h_{i-1}) \tag{12}$$

$$\phi_i = f_{out}(h_i) \tag{13}$$

where $\phi_i \in \mathbb{R}^m$ is the i-th step probabilistic output. Furthermore, the step loss L_i is a binary cross entropy (BCE) between the predicted outputs and the ground truth labels, which is formulated in the below equation:

$$\mathrm{BCE}(\phi_i, s_i) = -\frac{1}{m}\sum_{k=1}^{m}(s_i[k]\log\phi_i[k] + (1 - s_i[k])\log(1 - \phi_i[k])) \tag{14}$$

The whole network, including the Graph Feature Learning Network and the RNN, is trained in an end-to-end manner. Algorithm 1 summarizes the training procedure of our SCGG model.

Algorithm 1 Training Algorithm of SCGG Model

Input: Dataset of training graphs $\mathcal{D}$, number of new nodes m
Output: Learned functions f_{emb}, f_{RNN}, and f_{out}
for $\forall G \in \mathcal{D}$ **do**
  Build G_0 and G' from the graph G
end for
for number of training iterations **do**
  for $\forall G \in \mathcal{D}$ **do**
    $R = [r_1, r_2, \cdots, r_n] = f_{emb}(G_0)$
    $R' = [r'_1, r'_2, \cdots, r'_n, r'_{n+1}, \cdots, r'_{n+m}] = f_{emb}(G')$
    $R'' = [r''_i]_{i=1}^{n+m} = [r_1, r_2, \cdots, r_n, r'_{n+1}, \cdots, r'_{n+m}]$
    s_0 = sos; Initialize h_0; $L = 0$
    for i from 1 to $n + m$ **do**
      $x_i = Concat(r''_i, s_{i-1})$
      $h_i = f_{RNN}(x_i, h_{i-1})$
      $\phi_i = f_{out}(h_i)$
      $L = L + \mathrm{BCE}(\phi_i, s_i)$
    end for
    $L = L / (n + m)$
    Update model parameters by performing backpropagation to minimize the loss function L
  end for
end for

An example showing the SCGG model at training time is presented in Figs 3 and 4, where the graph of Fig 2 is used as training data. First, the representations of the nodes in both G_0

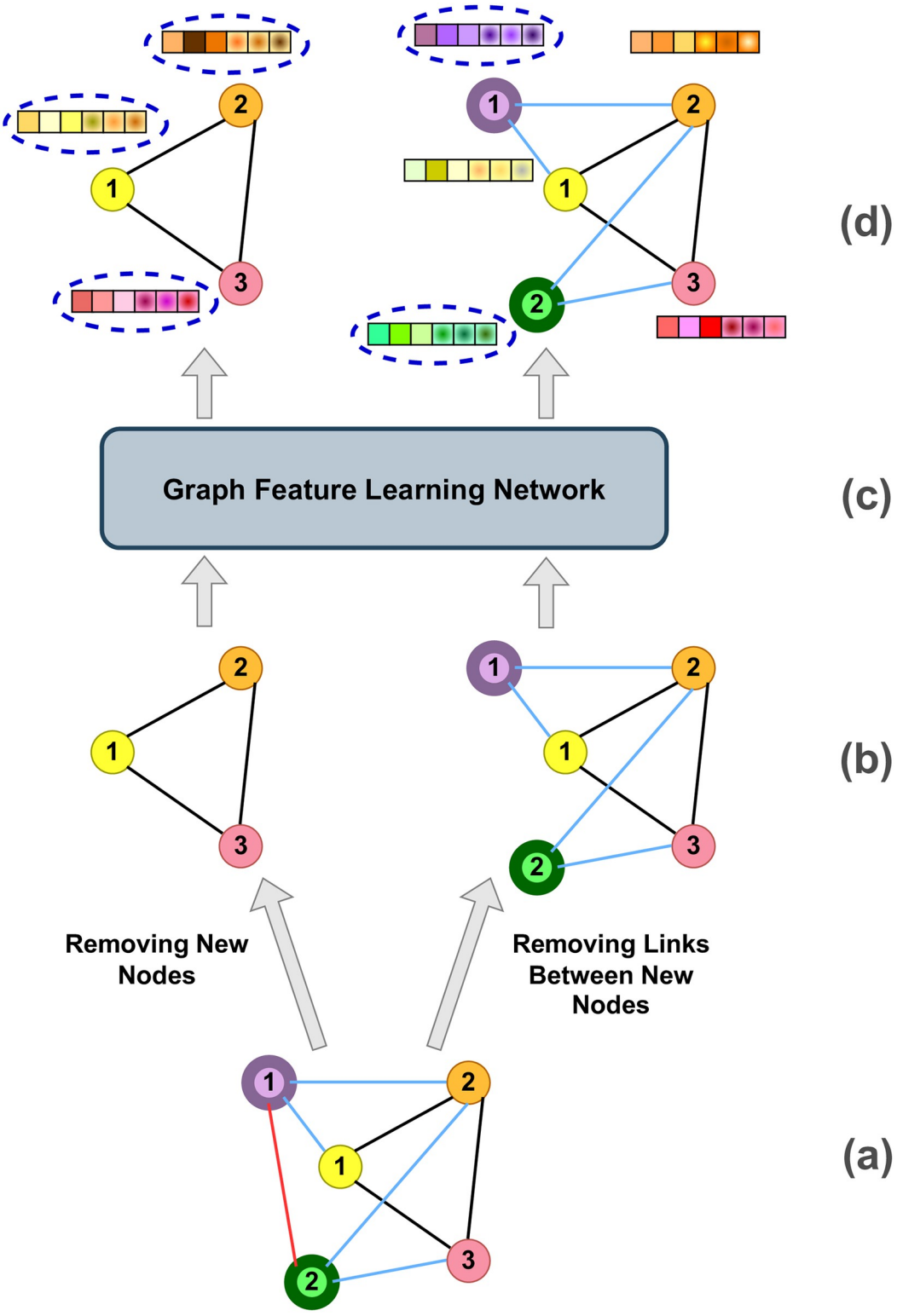

Fig 3. An overview of the workflow employed to obtain the required nodes' features in the training phase. (a) An input training graph after applying the preparation procedure shown in Fig 2(b) Two versions are made from the main graph. The one on the left will be treated as the initial graph (i.e., the G_0), and the graph on the right, which we denote in the paper by G', is obtained from the original graph by removing the intra-connection between the new nodes, i.e., the red link. (c) The Graph Feature Learning Network, whose architecture is illustrated in detail in Fig 1. (d) The features computed for each node of the graphs. The ones around which blue dashed ovals are drawn will be further used by the RNN.

https://doi.org/10.1371/journal.pone.0277887.g003

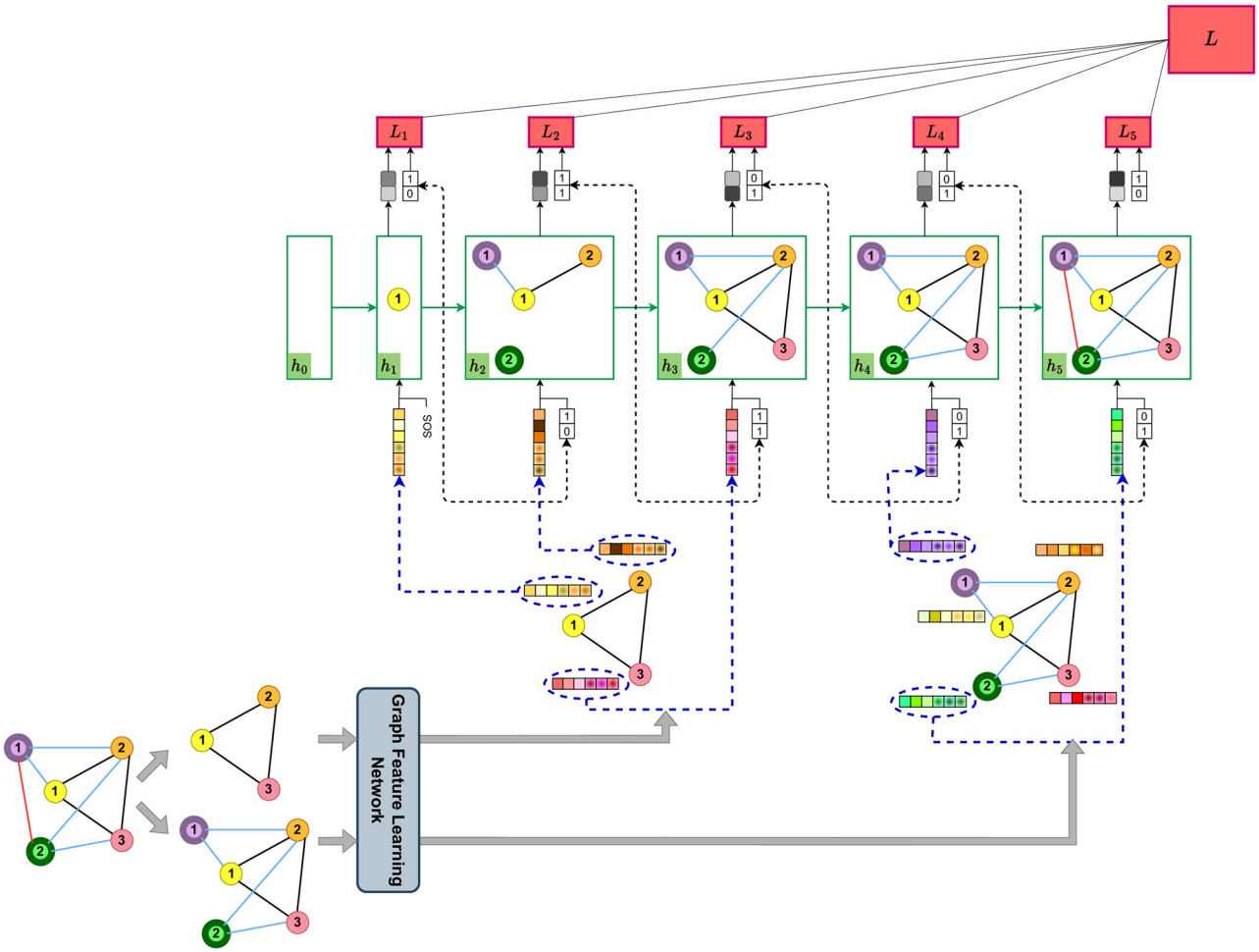

Fig 4. An example of the SCGG model at training time. For each graph node, including those in the initial graph (i.e., the G_0) and the ones in the set of new nodes (i.e., the $\tilde{V}$), the model outputs a probability distribution of link existence between that node and each new node (grey squares depict the probabilistic outputs, and the darker the colors, the higher the probabilities). To do this, at each step, a recurrent unit takes the features computed for one of the graph nodes (see Fig 3), as well as the previous node's true connections and the hidden state of the previous recurrent unit. In this regard, the nodes of G_0 (ordered by π_n) are first fed into the model, followed by the new nodes (ordered by π_m). Thus, the model learns to first generate the inter-links between the new nodes and those of G_0, and then predict the intra-links between the new nodes. The parameters of both the Graph Feature Learning Network and the RNN are updated by minimizing the total loss L that is obtained via aggregating the step losses L_i.

https://doi.org/10.1371/journal.pone.0277887.g004

and G' are computed by the Graph Feature Learning Network, which is illustrated in Fig 3. Then, the obtained representations for the G_0's nodes (see the left half of Fig 3(d)) are given to the RNN in the order specified by π_n. Accordingly, as depicted in Fig 4, in the first RNN step, it is the turn of node 1 (indicated by a yellow circle) to be processed, and thus its features are passed on to the first recurrent unit. The network then estimates the conditional probability distribution $P(\tilde{N}_1^{\pi_n}|N_1^{\pi_n}, N_2^{\pi_n}, N_3^{\pi_n})$, i.e., the probability of connecting the yellow node to each of the new nodes (the green and the purple ones). Afterwards, the step loss is calculated by taking the network output and the true labels (the first label is 1 because the yellow and the purple nodes are connected, and the second label is 0 because there is no edge between the yellow and the green nodes). In the second step, the second node's features (indicated by orange color), along with the true labels of the previous (yellow) node and the previous hidden state, are given to the recurrent cell. Then the network outputs an estimation of the $P(\tilde{N}_2^{\pi_n}|N_1^{\pi_n}, N_2^{\pi_n}, N_3^{\pi_n}, \tilde{N}_1^{\pi_n})$. The same procedure continues until all nodes of G, including the

ones in G_0 and the set of new nodes (i.e., $\tilde{V}$), are fed into the network. Thus, in the third step, the network outputs the probability of $P(\tilde{N}_3^{\pi_n}|N_1^{\pi_n}, N_2^{\pi_n}, N_3^{\pi_n}, \tilde{N}_1^{\pi_n}, \tilde{N}_2^{\pi_n})$ by taking into account the features of the third (pink) node in graph G_0. In the subsequent step, when all the initial graph's nodes have been processed, it is time to go through the new nodes in the order specified by π_m. Thus, the features computed for the first new node (displayed in purple color in the right half of Fig 3(d)) is given to the RNN to generate the probability $P(\tilde{M}_1^{\pi_m}|N_1^{\pi_n}, N_2^{\pi_n}, N_3^{\pi_n}, \tilde{N}_1^{\pi_n}, \tilde{N}_2^{\pi_n}, \tilde{N}_3^{\pi_n})$. Next, in the fifth step, the second new node's features (indicated in green) are fed into the recurrent network to produce the probability distribution $P(\tilde{M}_2^{\pi_m}|N_1^{\pi_n}, N_2^{\pi_n}, N_3^{\pi_n}, \tilde{N}_1^{\pi_n}, \tilde{N}_2^{\pi_n}, \tilde{N}_3^{\pi_n}, \tilde{M}_1^{\pi_m})$.

To elaborate a bit more on Fig 4, it is worth mentioning that each step's hidden state contains the information of a subgraph of the main graph (i.e., G). This subgraph includes the already processed graph nodes, the links connecting them, and their connections to each new node. It also includes links between the current node and the previous ones. For example, in Fig 4, in the third training step, two nodes (i.e., the yellow and the orange ones) have been processed and the pink node's features are fed into the recurrent unit as part of its input. Hence, the hidden state h_3 maintains a subgraph containing the link between the yellow and the orange nodes as well as the links between these nodes and the new nodes (shown by blue lines). It also retains the links between the current (pink) node and both the yellow and orange ones that have been fed into the network in the first two steps.

4.5 Inference

In the inference stage, an initial graph G_0 is given as the structural condition. Then, using the learned functions f_{emb}, f_{RNN}, and f_{out}, the model starts generating graph G by adding new nodes to G_0 and predicting the inter-links between the new nodes and those of G_0, as well as the intra-links between the new nodes themselves. Algorithm 2 describes the steps of the SCGG model at inference time. Moreover, Fig 5 illustrates the inference workflow of the SCGG by a toy example.

Algorithm 2 Inference Algorithm of SCGG Model

Input: f_{emb}, f_{RNN}, f_{out}, m, G_0
Output: G
$R = [r_1, r_2, \cdots, r_n] = f_{emb}(G_0)$
s_0 = sos; Initialize h_0
for i from 1 to n **do**
  $x_i = Concat(r_i, s_{i-1})$
  $h_i = f_{RNN}(x_i, h_{i-1})$
  $\phi_i = f_{out}(h_i)$
  $s_i \sim \phi_i$ ▷ Sample the inter-connections between the i-th node of G_0 and the set of new nodes
end for
Construct graph G' on top of G_0 using the sampled links $[s_1, s_2, \cdots, s_n]$
$R' = [r'_1, r'_2, \cdots, r'_n, r'_{n+1}, \cdots, r'_{n+m}] = f_{emb}(G')$
for j from $n + 1$ to $n + m$ **do**
  $x_j = Concat(r'_j, s_{j-1})$
  $h_j = f_{RNN}(x_j, h_{j-1})$
  $\phi_j = f_{out}(h_j)$
  $s_j \sim \phi_j$ ▷ Sample the intra-connections between the $j - n$-th new node and each of the new nodes
end for
Construct graph G on top of G' using the sampled links $[s_{n+1}, s_{n+2}, \cdots, s_{n+m}]$

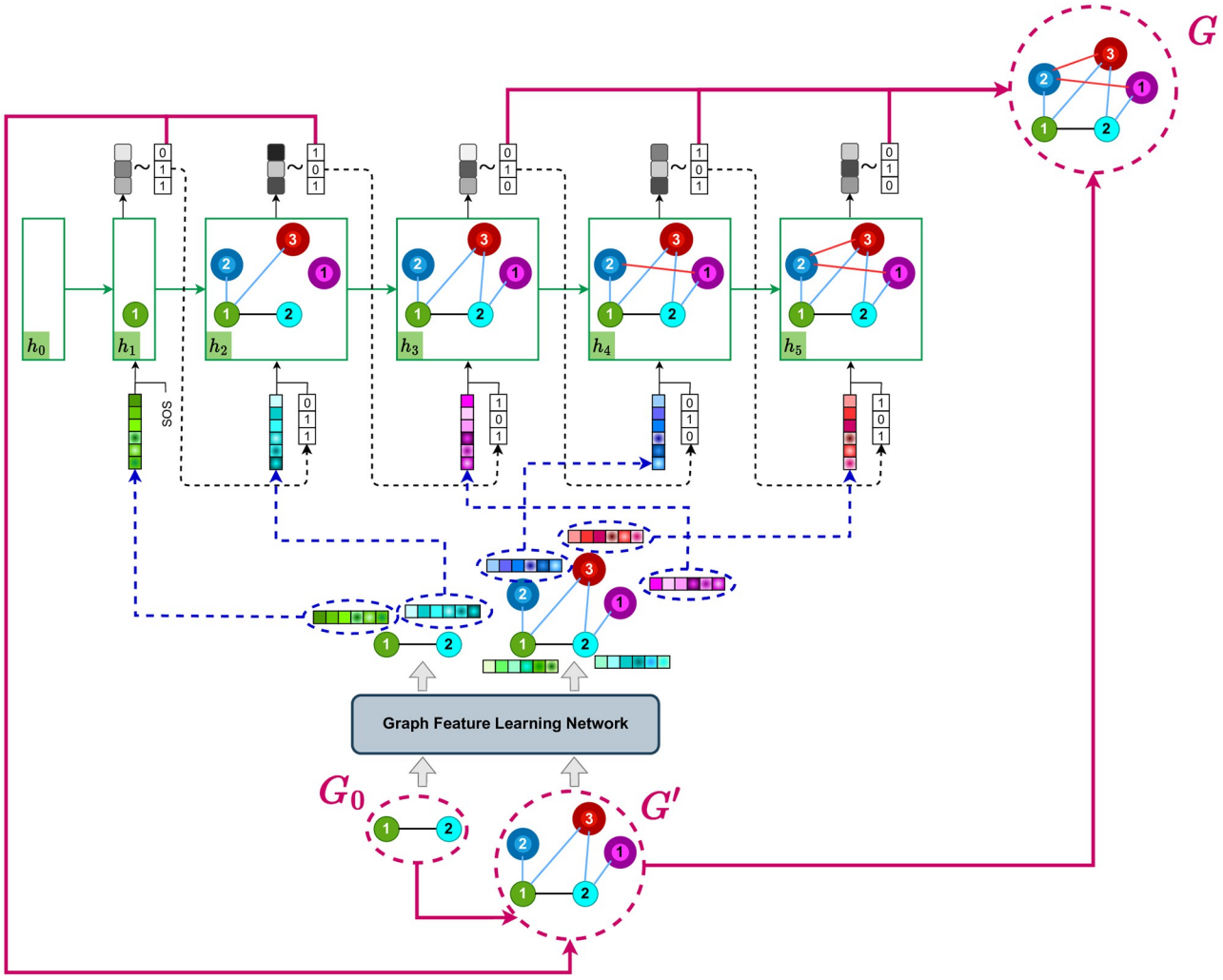

Fig 5. An example illustrating the SCGG model at inference time. In this example, $m = 3$ and a graph G_0 consisting of two nodes is given to the model as the structural condition. At first, the Graph Feature Learning Network computes representations for the G_0's nodes, which are then used as part of the RNN input. Next, the RNN proceeds for two steps and outputs the probabilities of the inter-connections between these two nodes and each new node. Therefore, all the inter-links are generated by sampling from the produced probabilities. At this point, it is time to construct graph G' based on the G_0 and the generated links. Next, G' is passed into the Graph Feature Learning Network to calculate the representations of its nodes. In this step, the representations of the new nodes are given to the RNN one by one to generate the intra-connections. Finally, the graph G is constructed on top of the G' by considering the generated intra-links.

https://doi.org/10.1371/journal.pone.0277887.g005

4.6 Implementation details

The proposed model is implemented using the PyTorch Library [62]. As previously discussed, the function f_{emb} consists of a graph convolutional network (GCN) and a Transformer network. In this regard, we use a two-layer GCN with the embedding size of each layer equaling 16. ReLU activation followed by a batch normalization layer are used between the two GCN layers. Besides, our Transformer has one encoder layer with 8 attention heads and a dropout of 0.1. We use 4 layers of GRU cells with a 128-dimensional hidden state to implement the function f_{RNN}. For the function f_{out}, a two-layer multilayer perceptron (MLP) is employed with 64 hidden units in the middle and a ReLU nonlinearity between the layers. Further, the Adam optimizer is used with a learning rate of 0.003, and the model is trained for 100 epochs with a

minibatch size of 32. Moreover, for the choice of π_n and π_m, we use uniform random orderings to maximize an approximation of the marginal likelihood in Eq 5, which becomes intractable to compute exactly as the size of graphs increases. To see details regarding hyperparameter tuning, please refer to S1 File.

5 Experiments

In this section, we first elaborate on both the synthetic and the real-world datasets we used for evaluation purposes. Then, we outline the state-of-the-art baselines with which we compare our SCGG model. Next, the evaluation metric is explained, followed by describing the experimental setup. Finally, we discuss the results of our proposed approach, as well as the ones of the competitor methods.

5.1 Datasets

We evaluate the performance of our proposed method on a variety of synthetic and real-world datasets. In the following, we provide a brief description of each dataset. Moreover, Table 2 summarizes the key statistics of them.

- **Grid**: It is a synthetic dataset consisting of standard 2D grid graphs.
- **IMDBBINARY**: This dataset consists of ego-networks derived from actor/actress collaborations based on the information of movies belonging to the Action and Romance genres on IMDB. For each graph, nodes represent actors/actresses, and if a pair of them appears in the same movie, a link connects their corresponding nodes in the graph.
- **IMDBMULTI**: The same explanation given for the IMDBBINARY dataset is valid for this dataset as well, except that the movies belong to the Comedy, Romance, and Sci-Fi genres.
- **Enzymes**: This dataset consists of graphs each representing a protein tertiary structure from the BRENDA enzyme database [63]. More precisely, a graph's nodes represent secondary structure elements (SSEs). An edge connects two nodes if their corresponding SSEs are neighbors along the amino acid sequence or one of the three nearest neighbors in space.
- **NCI1**: It is a biological graph dataset published by the National Cancer Institute (NCI). Each graph in the dataset represents a chemical compound screened for its activity against the growth of human tumors.
- **Protein**: This dataset contains protein graphs [64]. Each graph represents a protein with nodes corresponding to amino acids. If the distance between two amino acids of a protein is less than 6 Angstroms, their corresponding nodes are connected in the graph.

Table 2. Statistics of datasets used in the experiments.

Dataset	Min. # Nodes	Max. # Nodes	Avg. # Nodes	Std. # Nodes	Avg. Sparsity	# Graphs
Grid	25	361	144	75.83	0.97	225
IMDBBINARY	12	136	19.77	10.06	0.51	1000
IMDBMULTI	11	89	18.83	9.75	0.44	686
Enzymes	15	125	34.61	13.86	0.87	545
NCI1	11	111	30.05	13.48	0.92	4075
Protein	100	500	257.87	105.50	0.98	918

https://doi.org/10.1371/journal.pone.0277887.t002

5.2 State-of-the-art approaches

We compare our approach with several well-known state-of-the-art methods, explanations of which are provided in the following.

- **KronEM** [52]. This is an old and well-known network completion method that combines the Expectation-Maximization (EM) framework with the Kronecker graphs model [65] to infer missing nodes and their corresponding edges in partially observed graphs. To do this, in each EM iteration, the method first utilizes the observed part of a graph to estimate model parameters (the M-step). Then it infers the missing part of that graph using the estimated model (the E-step).
- **GraphRNN-S** [32]. This is a very famous autoregressive deep graph generator that first transforms graphs into sequences and then models the corresponding data distribution using RNNs. At each step, the method adds a new node to the currently generated graph and predicts the links connecting it to the previous nodes. Aside from that, GraphRNN-S makes a simplistic assumption that a node's links are independent of each other, and therefore models them by a multi-layer perceptron.
- **GraphRNN** [32]. This is the full GraphRNN model, which is relatively similar to GraphRNN-S, with the difference that it does not take into account the edge independence as a simplifying assumption. Therefore, to capture the interdependencies between a node's edges, it employs another recurrent neural network called the edge-level RNN.
- **DeepNC** [56]. This is the most recent graph completion baseline that utilizes a deep generative model of graphs, namely GraphRNN-S, to infer the missing parts of a partially observable network. To this end, the method first learns a likelihood over data by training the GraphRNN-S model. Then, it proposes a sequence of algorithmic steps to recover the network in a greedy fashion, trying to maximize the learned likelihood. The fact needed to be noted here is that although this method uses the probabilities generated by a deep generative model of graphs to make algorithmic decisions, it is not considered a totally deep learning-based approach. However, if a model is specifically trained to address the problem of graph completion, it can achieve higher performance.
- **EvoGraph** [66]. This is a graph upscaling method, which expands an initial input graph $G_0 = (V_0, E_0)$ in K stages by adding $|E_0|$ new edges at each stage. The method considers a set of candidate new nodes in every expansion phase and adds each new edge by choosing one of its endpoints from the current nodes and the other from the candidate ones. To provide a fair comparison between EvoGraph and other methods, we make a slight change to its upscaling process by terminating it right after the insertion of the m-th new node.

5.3 Comparison of time complexities

In this subsection, and after explaining the baselines with which we compare our model, we analyze the computational complexity of these competing approaches. To begin, we examine the complexity of our model. As discussed earlier, SCGG predicts m probabilities at each inference stage to determine how the current node should connect to each new node. As there are $n + m$ inference steps, the total complexity of SCGG is $\mathcal{O}(m(n + m))$. In light of the fact that $m \ll n$, which DeepNC also considers, the complexity of SCGG can be rewritten as $\mathcal{O}(n)$. For the DeepNC approach, it is proved in [56] that the lower and upper bounds of its computational complexity are $\Omega(n)$ and $\mathcal{O}(n^2)$, respectively, which shows that DeepNC can be computationally more expensive than SCGG.

The complexity of both GraphRNN-S and GraphRNN, as stated by their authors, is $\mathcal{O}(M(n+m))$, which can be simplified into $\mathcal{O}(M(n))$ when considering $m \ll n$. Therefore, depending on the choice of M, which is typically estimated by an empirical upper bound, we can say that the complexity of SCGG is less than or equal to that of GraphRNN-S and GraphRNN.

According to what has been mentioned in the previous subsection, the slightly modified version of EvoGraph can generate at maximum $n(m-1)$ edges before the insertion of the m-th new node, which connects nodes in the initial graph to the new ones. Considering that each edge insertion operation is of $\mathcal{O}(1)$ complexity, the total computational complexity of EvoGraph is $\mathcal{O}(n(m-1)+1)$, which as discussed earlier, can be simplified as $\mathcal{O}(n)$. Therefore, this method is of comparable computational complexity to our SCGG model.

Finally, the runtime of kronEM, according to what is stated by its authors, scales as $\mathcal{O}(|E|\log|E|)$ with $|E|$ representing the number of edges in the final graph G. Assuming that |E| is at least of order $\mathcal{O}(n+m)$, which is the case in many graphs, including the ones in our datasets (please see S2 File for more details), the complexity of kronEM is at least $\mathcal{O}((n+m)\log(n+m))$. As a result, KronEM is more computationally expensive than our approach.

5.4 Evaluation metric

Similar to [56], we use *Graph Edit Distance (GED)* [67] as the evaluation metric to assess the performance of our SCGG method and the baselines. In this regard, if we denote a generated or completed graph by $\hat{G}$ and its corresponding ground truth graph by G, the GED between these two graphs, which shows how dissimilar they are, can be formulated as follows:

$$d(\hat{G}, G) = \min_{\lambda \in \gamma(\hat{G},G)} \sum_{e_i \in \lambda} c(e_i) \tag{15}$$

where $\gamma(\hat{G}, G)$ is the set of all edit paths converting $\hat{G}$ to a graph that is isomorphic to G. Moreover, $c(e_i)$ is the cost of an edit operation e_i, which in the same way as [56], we set it to 1 for all operations. Additionally, as with [56], we normalize the GED computed for each pair of graphs by the average of their sizes.

Along with our brief overview of GED, one important point to note is that enumerating all the discussed edit paths requires employing a combinatorial search procedure with exponential time complexity, and therefore the exact solution to this problem is NP-complete [68]. Hence, we utilize an approximation approach [69] for computing GED scores.

5.5 Experimental setup

In addition to what we have explained in Section 4.6 concerning the details of implementing our SCGG model, in this subsection, we elaborate on the remainder of the details regarding the experimental setup. In this respect, we select a random subset of 80% of the graphs in each dataset to train our model. A similar approach is also followed to train other learning-based baselines (i.e., GraphRNN-S and GraphRNN). We then make use of the remaining 20% of graphs for model testing. More specifically, for each graph G in the test set, we perform the following two steps for 10 iterations:

- We randomly choose a number of m nodes from the original test graph G and remove these nodes and their associated edges to acquire a subgraph G_0.
- We then feed the obtained subgraph to all the competing methods and compare their results to the ground truth graph G.

An illustrative example depicting our SCGG model at the evaluation time is provided in S3 File.

Afterwards, for each graph in the test data, we average the GED scores calculated in 10 iterations and compute their standard deviation. Finally, for each parameter value m, we report the average GED scores, as well as the average standard deviations computed over the whole test set.

5.6 Results and discussion

In this subsection, the experiments conducted to evaluate the performance of our proposed method against the baselines are presented in three parts. In the first part, we set the maximum possible value for the parameter m such that the competing methods can be evaluated on all datasets. Then, we compare the obtained results and report the gain of SCGG over the baselines. In the second part, we discretely change the value of m from the lowest to the highest possible amount so that all datasets can be utilized for model testing. Then we study how the performances of various methods are affected by increasing the value of m. Finally, in the third part, we raise m to much higher values and evaluate the efficacy of all approaches on the dataset that offers this possibility.

We first analyze the performance of different methods for the case where $m = 10$. The reason for choosing this value for m is that, as outlined in Table 2, the minimum number of nodes among graphs of all datasets is 11. Hence, to construct initial graphs G_0, a maximum of 10 nodes can be removed from the original graphs. We report the obtained results in Table 3, from which it is evident that for all datasets, SCGG is the best-performing method in terms of the lowest average GED score. More precisely, SCGG obtains an average gain of 51.74% over other approaches based on the experiments conducted on all datasets, with the lowest gain value of 2.65% and the highest gain of 88.15%. Furthermore, in most cases, the standard deviations of our results are less than those of the baselines.

Besides, the results of Table 3 reveal that KronEM does not perform well in general, so that, unlike other methods, its average GED has never been lower than 0.52. There can be several reasons for this. First, unlike SCGG, GraphRNN-S, GraphRNN, and to some extent DeepNC, this method is not trained on a dataset of graphs, but rather, it processes each graph in the test set separately, i.e., it completes the structure of each partially observed graph based solely on

Table 3. Comparison of SCGG with its competitors for $m = 10$ in terms of GED (Avg. ± Std.).

Method \ Dataset		Grid	IMDBBINARY	IMDBMULTI	Enzymes	NCI1	Protein
KronEM (Y_1)		0.5875 ± 0.1900	0.5634 ± 0.2065	0.5650 ± 0.2068	0.5698 ± 0.1952	0.6521 ± 0.1639	0.5281 ± 0.1812
EvoGraph (Y_2)		0.2181 ± 0.0170	0.6831 ± 0.1467	0.6755 ± 0.1374	0.5274 ± 0.0582	0.4695 ± 0.1597	0.0890 ± 0.0082
GraphRNN-S (Y_3)		0.1460 ± 0.0580	0.4338 ± 0.2094	0.4116 ± 0.2001	0.6962 ± 0.1641	0.7666 ± 0.1146	0.0898 ± 0.0732
GraphRNN (Y_4)		0.1293 ± 0.0622	0.4620 ± 0.2304	0.4620 ± 0.2264	0.6802 ± 0.1689	0.7600 ± 0.1219	0.0930 ± 0.0740
DeepNC (Y_5)		0.4830 ± 0.0657	0.2984 ± 0.1203	0.3049 ± 0.1365	0.5525 ± 0.2005	0.6642 ± 0.1505	0.1355 ± 0.0177
SCGG (X)		0.0701 ± 0.0232	0.2905 ± 0.1129	0.2871 ± 0.1083	0.2046 ± 0.0553	0.2688 ± 0.0671	0.0626 ± 0.0135
Gain	$\frac{Y_1-X}{Y_1} \times 100$	88.07	48.44	49.19	64.09	58.78	88.15
	$\frac{Y_2-X}{Y_2} \times 100$	67.86	57.47	57.50	61.21	42.75	29.66
	$\frac{Y_3-X}{Y_3} \times 100$	51.99	33.03	30.25	70.61	64.94	30.29
	$\frac{Y_4-X}{Y_4} \times 100$	45.78	37.12	37.86	69.66	64.63	32.69
	$\frac{Y_5-X}{Y_5} \times 100$	85.49	2.65	5.84	62.97	59.53	53.80

https://doi.org/10.1371/journal.pone.0277887.t003

the available part of it. Another reason for the underperformance of KronEM might be due to the fact that the Kronecker graphs model generates graphs with 2^x nodes. Therefore, when an initial graph G_0 is given to KronEM, it increases the number of its nodes to the nearest power of 2. This can lead to a significant difference between the ground truth and the completed graph regarding the number of nodes, thereby causing the GED score to be raised.

In addition to what we have discussed so far regarding the results in Table 3, they also indicate that EvoGraph considerably underperforms on the IMDBBINARY and IMDBMULTI datasets. This is because the upscaling process of EvoGraph tends to establish connections with new nodes that have not yet been linked to the graph. In other words, adding new edges is performed with a high priority to connect new nodes to the already generated graph, meaning that setting up more connections between the previously added nodes and the nodes of the initial graph G_0 is carried out with a relatively low priority. Thus, it is not surprising that the graphs produced by EvoGraph generally contain fewer edges than the ones belonging to the IMDBBINARY or IMDBMULTI datasets, which according to the statistics listed in Table 2, have low edge sparsity. In light of this, we can expect a decrease in the performance of EvoGraph on these two datasets.

In the second part of the experiments, we vary the value of m discretely from 1 to 10 and study the performance of different methods as a function of the parameter m. In this regard, Figs 6–11 demonstrate the obtained results on the Grid, IMDBBINARY, IMDBMULTI,

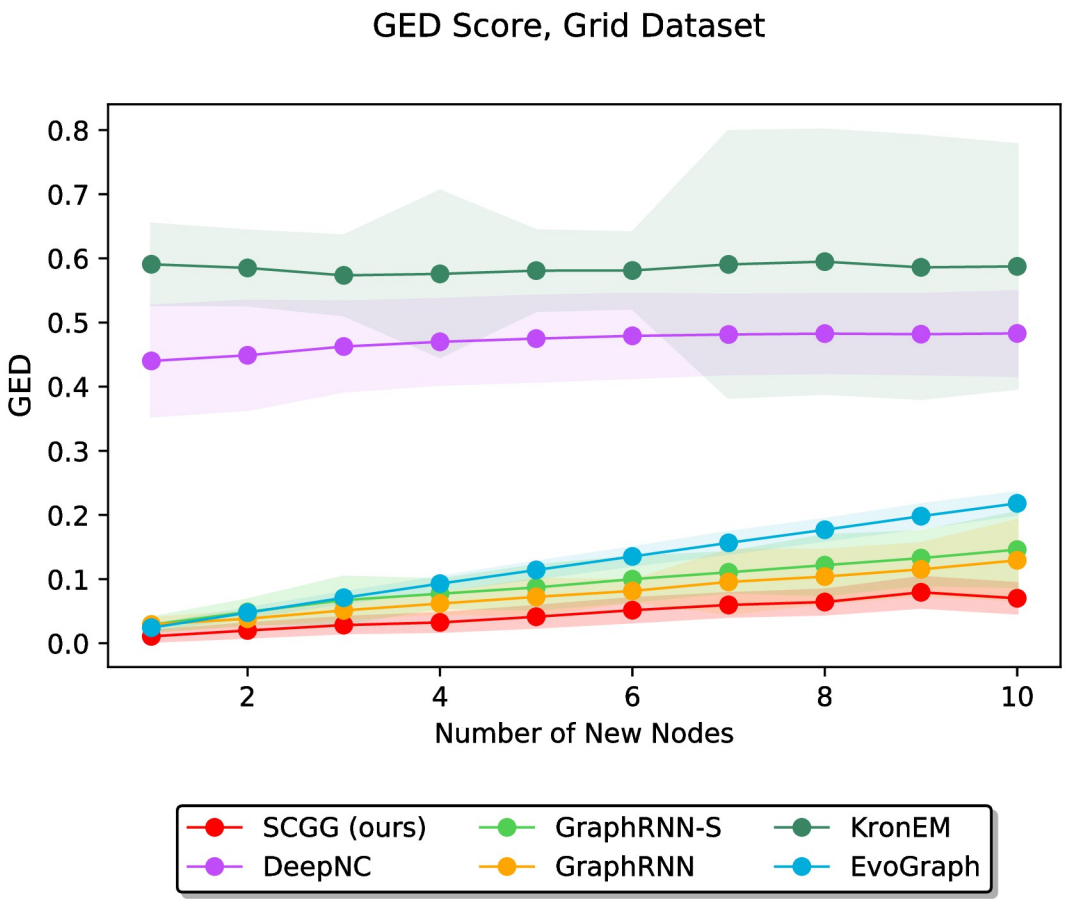

Fig 6. Performance comparison on the Grid dataset in terms of GED (lower is better) as a function of the number of new nodes to be added (i.e., m).

https://doi.org/10.1371/journal.pone.0277887.g006

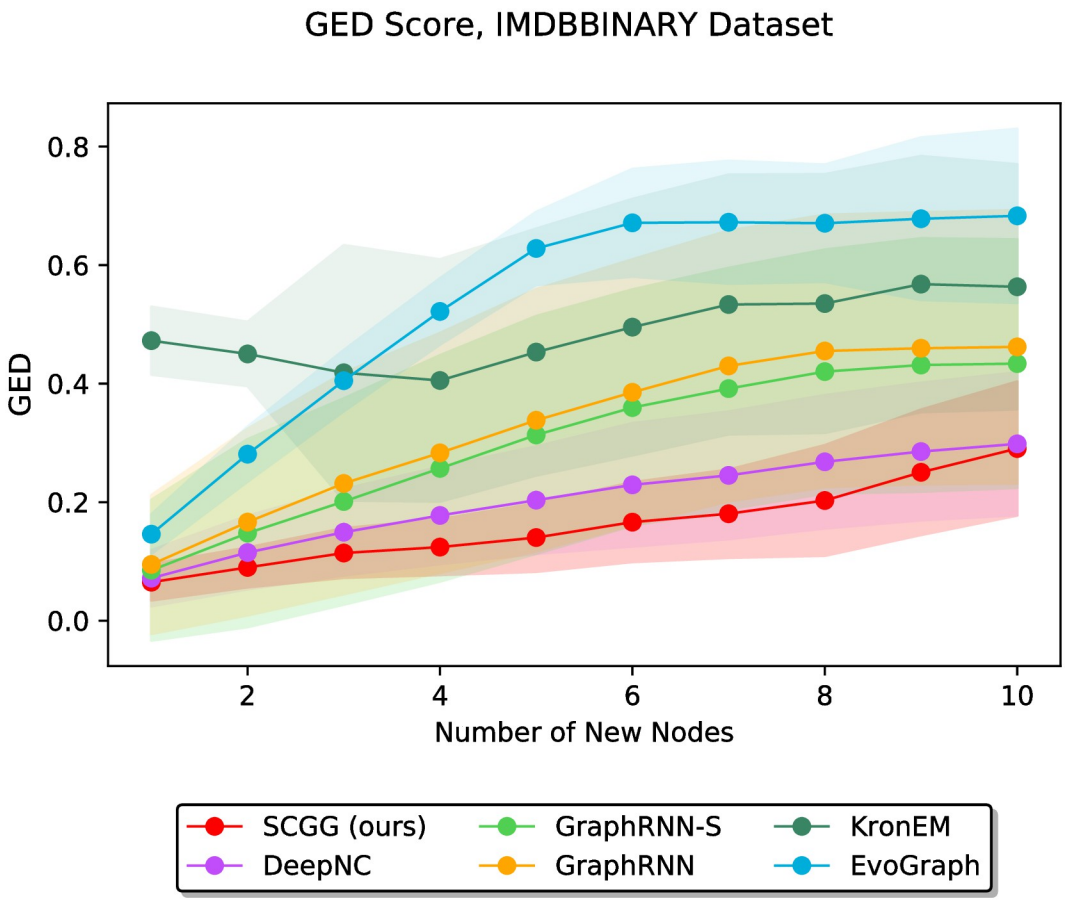

Fig 7. Performance comparison on the IMDBBINARY dataset in terms of GED (lower is better) as a function of the number of new nodes to be added (i.e., *m*).

https://doi.org/10.1371/journal.pone.0277887.g007

Enzymes, NCI1, and Protein datasets, respectively. Moreover, since a part of the results are somewhat visually overlapped, which may affect their readability, we provide the readers with another view of them. In this regard, a pairwise comparison between our SCGG approach and each of the baselines is depicted in a separate subplot for all datasets. Accordingly, the second appearance of the results in Figs 6–11 can be found in S1–S6 Figs, respectively. In the following, we discuss the results obtained on each dataset.

Fig 6 shows the effect of increasing the value of m on the performance of various methods on the Grid dataset. According to these results, the GED values of most methods (i.e., SCGG, GraphRNN-S, GraphRNN, and EvoGraph) increase almost uniformly with the growth of m, which makes sense since as m increases, the task becomes more difficult. A noteworthy point here is that our proposed SCGG approach performs the best (lowest GED score). In addition, as m gets higher values, the GED of our approach increases with a lower slope. This figure also demonstrates the poor performance of KronEM (both in terms of the relatively high average GED score and the high standard deviations), which is in accordance with what we discussed before. The results also indicate that DeepNC underperforms on the Grid dataset. This may be because DeepNC, unlike other competitors, does not conduct its processing steps by taking into account the whole initial graph G_0 at once. To put it another way, other methods receive an initial graph G_0 and start adding new nodes on top of it. Meanwhile, DeepNC starts

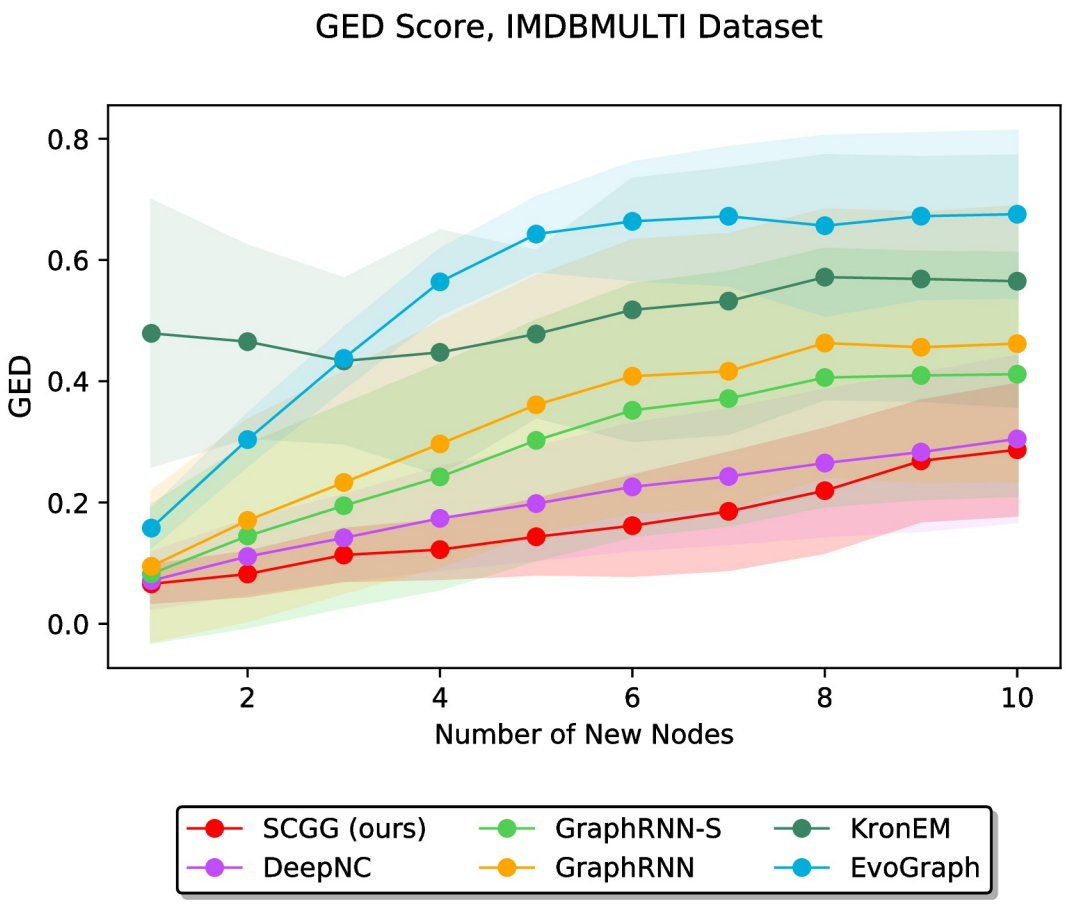

Fig 8. Performance comparison on the IMDBMULTI dataset in terms of GED (lower is better) as a function of the number of new nodes to be added (i.e., *m*).

https://doi.org/10.1371/journal.pone.0277887.g008

constructing the graph from scratch, and at each stage, it randomly decides whether to choose the next node from the set of initial graph nodes or add a new one. Therefore, since the graphs of the Grid dataset follow a highly regular structural pattern, not considering complete information of initial graphs at once before processing can lead to the performance drop of DeepNC by constructing graphs that are substantially different from the expected ones.

Figs 7 and 8 show the results obtained on the IMDBBINARY and IMDBMULTI datasets, respectively. They reveal that for all values of m, the SCGG outperforms the baselines. It is also evident from these results that EvoGraph has achieved the worst performance among other competitors. This, as explained earlier, can be due to the tendency of EvoGraph to complete the graph structures by adding a small number of edges to G_0, which is in contrast to the non-sparsity of the graphs belonging to these two datasets.

The results on datasets Enzymes and NCI1 are depicted in Figs 9 and 10, respectively. Since these two datasets share relatively similar statistical properties, as listed in Table 2, somewhat similar results are observed on them. In this regard, our SCGG approach achieves the best performance compared to other methods. Specifically, it offers the lowest average GED score in almost all cases. Moreover, in the vast majority of circumstances, the standard deviations of the results obtained by our method are lower compared to the other approaches. These results also demonstrate that GraphRNN-S and GraphRNN perform the worst as the value of m

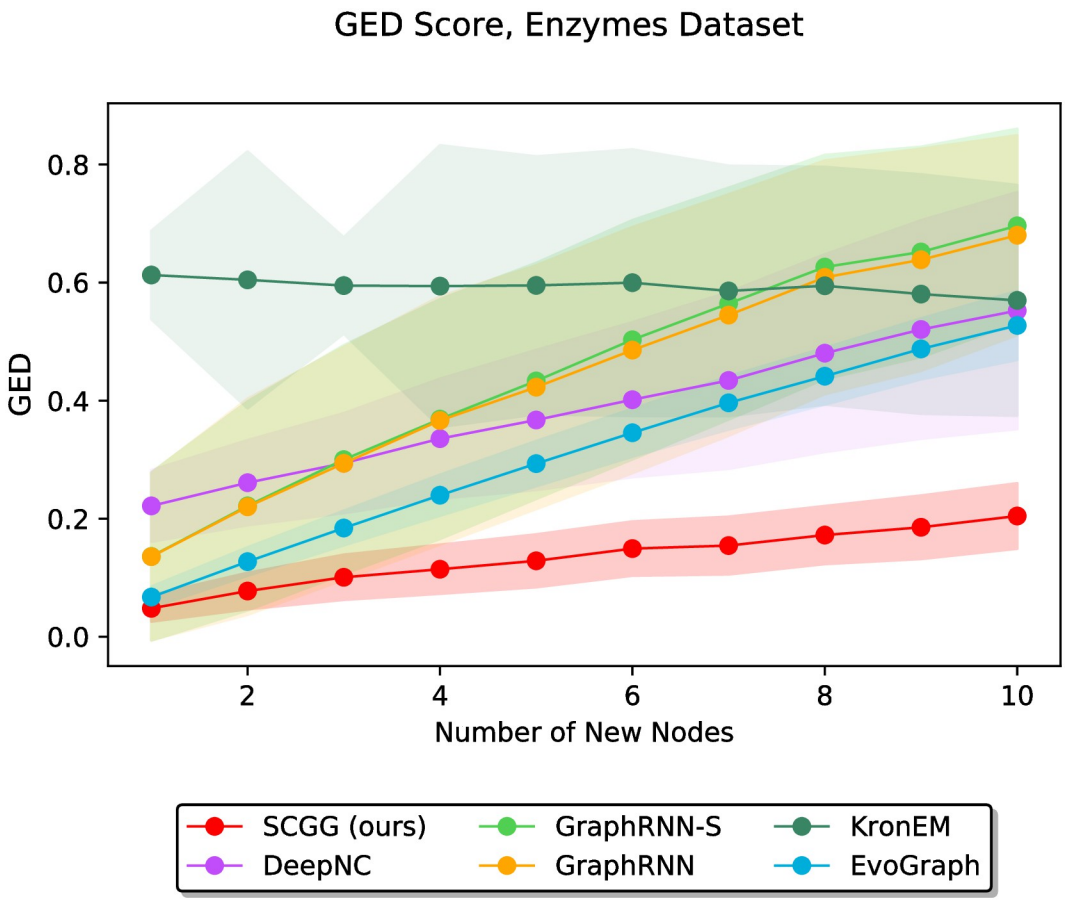

Fig 9. Performance comparison on the Enzymes dataset in terms of GED (lower is better) as a function of the number of new nodes to be added (i.e., *m*).

https://doi.org/10.1371/journal.pone.0277887.g009

increases. This is because these two are general graph generation approaches, which are not specifically designed to solve problems such as structure-conditioned graph generation or graph completion. Therefore, although they have achieved acceptable performance in some cases, it is not surprising that in some other cases, they perform poorly compared to the baselines.

Finally, Fig 11 depicts the results on the Protein dataset, in which the value of m varies discretely from 1 to 10. The results indicate that the SCGG method obtains a lower GED than the baseline methods in almost all cases, and as the value of m goes up, this performance superiority more clearly manifests itself. In addition, the weak performance of KronEM can be seen in these results, the reasons for which have been discussed in detail previously.

In the third part of the experiments, we study the performance of all competing approaches in the case where a much larger number of nodes are supposed to be added to initial graphs G_0. Accordingly, we conduct the experiments on the Protein dataset, which gives us this opportunity due to the large size of its graphs. More precisely, we increase the value of the parameter m from 10 to 90 (i.e., the maximum possible value that does not exceed the minimum number of nodes in this dataset) in steps of 10. The results of these experiments are illustrated in Fig 12. As we can see, our method achieves the best results in terms of the lowest GED score for all values of m. Furthermore, in most cases, and especially as m gets higher

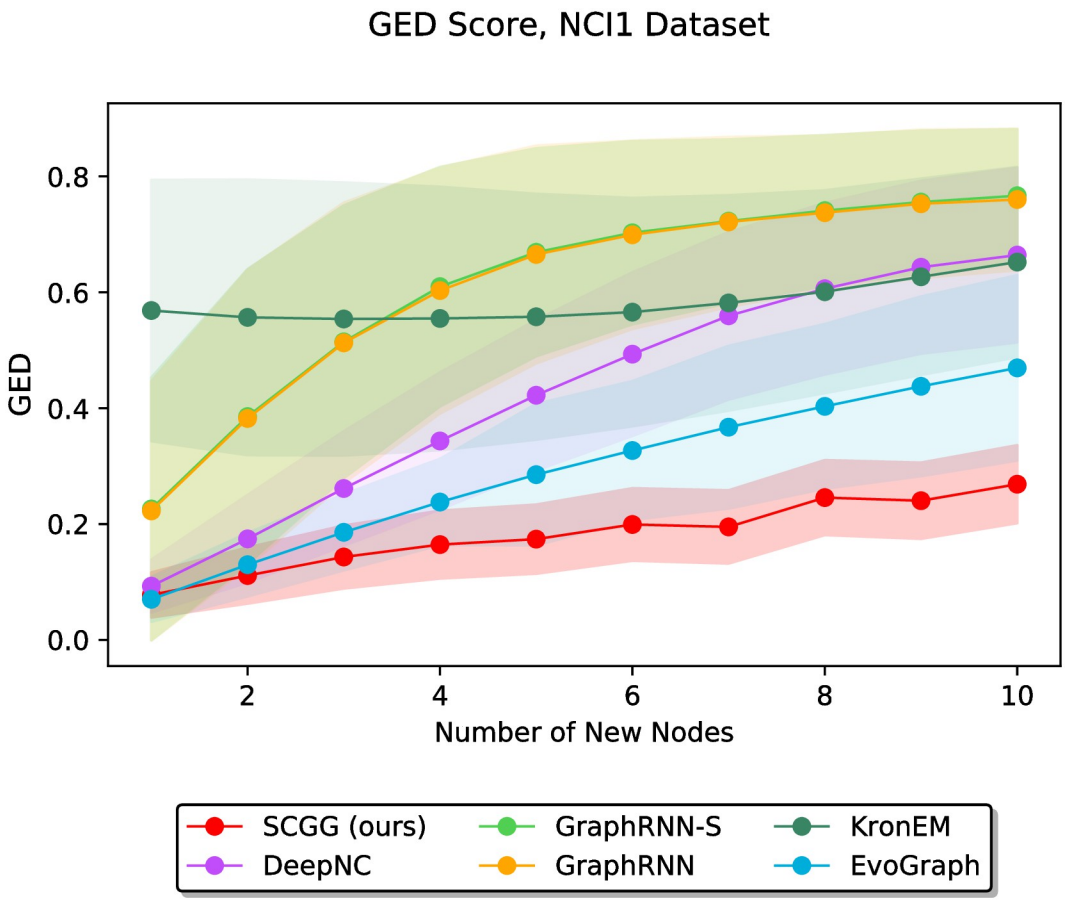

Fig 10. Performance comparison on the NCI1 dataset in terms of GED (lower is better) as a function of the number of new nodes to be added (i.e., *m*).

https://doi.org/10.1371/journal.pone.0277887.g010

values, our results show smaller standard deviations than those of other approaches. We can also observe that for the higher values of *m*, for which both the tasks of graph completion and structure-conditioned graph generation become much more challenging, the performances of GraphRNN-S, GraphRNN, and EvoGraph deteriorate rapidly. This can be interpreted according to the fact that these approaches are not particularly designed to address such tasks. Conversely, as the parameter *m* rises to its highest values, SCGG, DeepNC, and KronEM offer the best results, respectively.

Another perspective of the results in Fig 12 can be found in S7 Fig, providing the readers with a pairwise comparison of our SCGG model and each of the baselines.

As shown by the experimental results, our SCGG method outperforms its competitors in almost all cases. At the end of this subsection, we summarize the reasons for its better performance. In our opinion, this is due to the fact that our SCGG model is the first deep learning-based generative approach to exclusively address structure-conditioned graph generation, one of whose main applications is solving the graph completion problem. We have compared the performance of our proposed method with the best and most well-known baselines closely related to our research. These baselines can broadly be classified into three categories. The first category is modern graph generation approaches. GraphRNN-S and GraphRNN fall into this category. Despite the impressive results achieved by these two methods in terms of

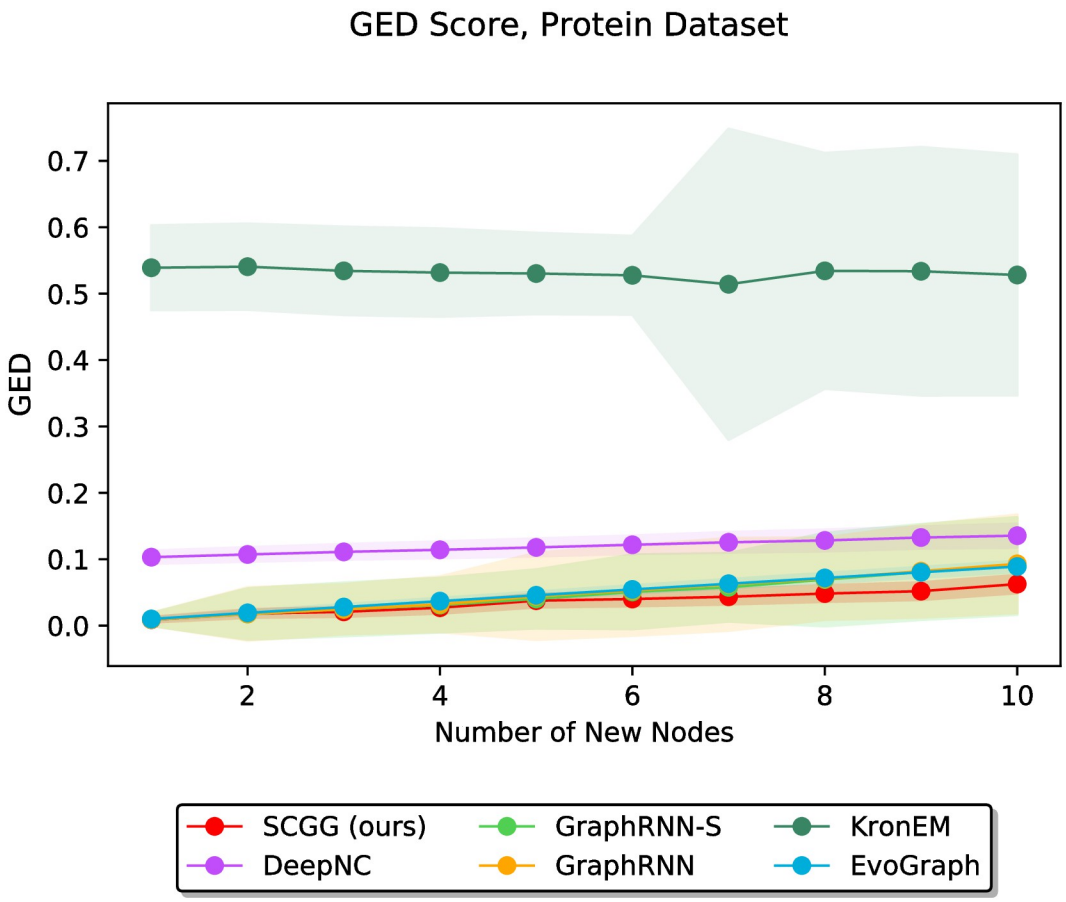

Fig 11. Performance comparison on the Protein dataset in terms of GED (lower is better) as a function of the number of new nodes to be added (i.e., *m*).

https://doi.org/10.1371/journal.pone.0277887.g011

distribution-related metrics for generating graphs in general, our method outperforms them for the task of structure-conditioned graph generation. This is primarily because the SCGG imposes structural conditions on the generation procedure mainly through its Graph Feature Learning Network, whereas GraphRNN-S and GraphRNN do not include such mechanisms. The second category of baselines relates to graph completion methods. The KronEM and DeepNC approaches fall under this category, where KronEM processes each graph separately. In more precise terms, KronEM predicts the missing nodes of a graph solely based on the observed parts of that graph. Contrary to this, our approach is based on learning from graph datasets, enabling it to solve the graph completion problem more efficiently. Another limitation significantly affecting the performance of KronEM is the number of nodes in its resulting graphs, which needs to be a power of two. The second graph completion baseline with which we have compared our model is the recently proposed DeepNC approach. This method, as mentioned earlier, first trains a well-known graph generative model (i.e., the GraphRNN-S) and then proposes a set of algorithms to complete partially observed graphs in a manner that maximizes the likelihood of the trained model. Despite the innovative algorithms and promising results of DeepNC against its competitors, it is not a learning-based approach specifically trained for graph completion, which could be the key reason for its inferior performance compared with SCGG. Lastly, we have compared our method to EvoGraph, which falls under the

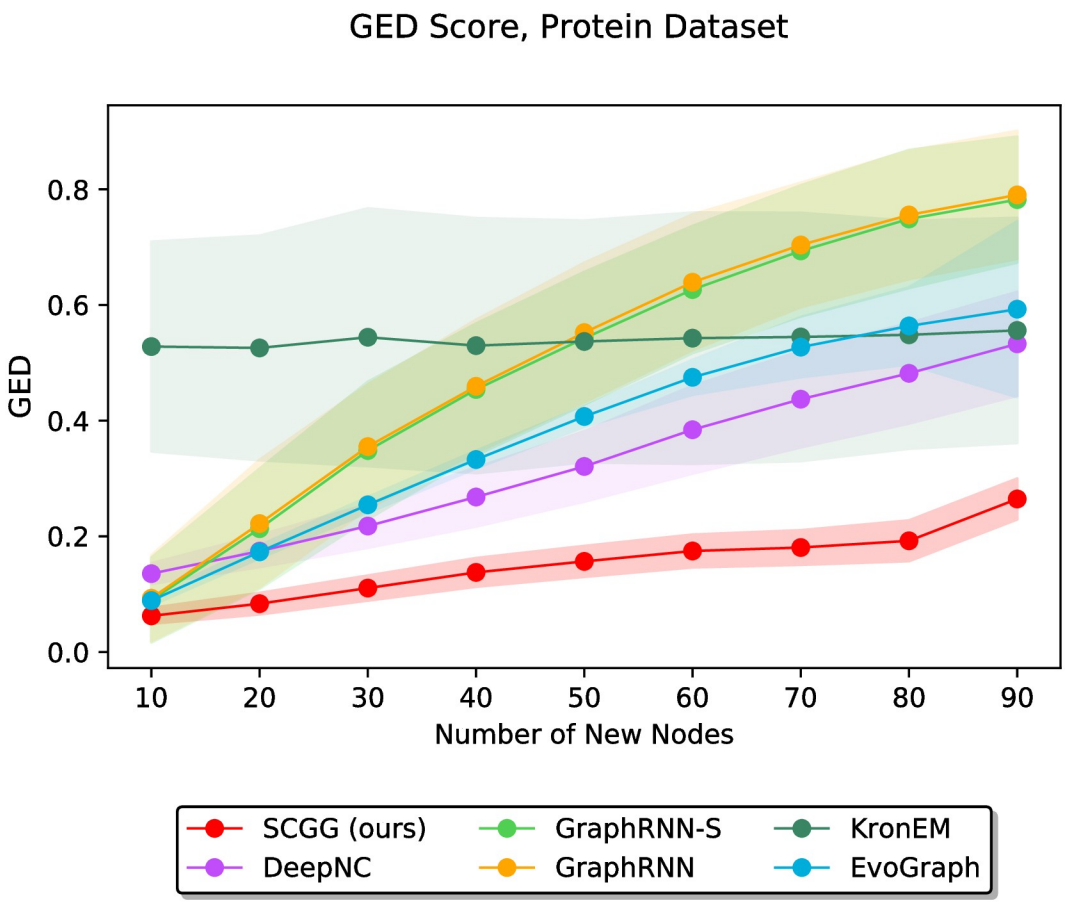

Fig 12. Performance comparison on the Protein dataset in terms of GED (lower is better) as a function of the parameter *m*, which varies discretely from 10 to 90 in steps of 10.

https://doi.org/10.1371/journal.pone.0277887.g012

category of graph upscaling methods. As with the DeepNC approach, this method adopts an algorithmic strategy without being trained on graph datasets, which results in its poorer performance than the SCGG method.

6 Conclusions

In this work, we have presented SCGG, a novel structure-conditioned graph generation approach that autoregressively generates a graph by adding new nodes and their corresponding edges on top of a given initial substructure G_0. Specifically, the architecture of our model consists of a specific graph representation learning network, which is the main responsible for considering the conditioning substructure, and an autoregressive generative model (i.e., a recurrent neural network) that mostly maintains the generation history. We then have employed this model to address the intrinsically hard-to-solve problem of network completion, in which the goal is to complete the structure of a partially observed graph, some of whose nodes are unknown. To demonstrate the superiority of our proposed SCGG model, we have conducted intensive experiments on both synthetic and real-world datasets and compared the performance of our method against state-of-the-art baselines for the task of graph completion. The experimental results illustrate that SCGG outperforms the baselines in terms of the GED score, which indicates that the graphs generated by our model, on average, are the

closest to the ground truth graphs. To the best of our knowledge, this is the first time a completely deep learning-based approach addresses the graph completion problem.

The present study does not address all the problems regarding structure-conditioned graph generation. First, our SCGG model tackles a particular type of condition by generating several new nodes and their corresponding edges on top of a given conditioning substructure. It is possible, however, to envisage other structural conditions. For example, in some situations, it may be desirable that structure-related statistical properties of generated graphs (e.g., average node degree, average clustering coefficient) be in a specific value range. Secondly, as with many other graph generation approaches, our method decides based on the structural information of graphs (i.e., the way nodes are connected), and does not take into account different types of graphs or additional information associated with them. There are, however, many types of graphs. For example, heterogeneous graphs with multi-typed nodes and edges are widely common in real-life. Molecular graphs are another example with significant applications in the pharmaceutical industry. Lastly, all the current graph generation techniques, including SCGG, suffer from scalability issues, making them inapplicable for graphs with millions or even tens of thousands of nodes. Here, a question arises as to whether we can transcend these limitations. As future research pathways, we intend to investigate the answer by further expanding the framework established by SCGG.

Supporting information

S1 Fig. Pairwise performance comparison between our proposed SCGG method and its competitors on the Grid dataset. The results are reported in terms of GED (the lower the better) as a function of the number of new nodes (denoted by m) that are added to initial graphs (each represented by the notation G_0 in the paper).
(TIF)

S2 Fig. Pairwise performance comparison between our proposed SCGG method and its competitors on the IMDBBINARY dataset. The results are reported in terms of GED (the lower the better) as a function of the number of new nodes (denoted by m) that are added to initial graphs (each represented by the notation G_0 in the paper).
(TIF)

S3 Fig. Pairwise performance comparison between our proposed SCGG method and its competitors on the IMDBMULTI dataset. The results are reported in terms of GED (the lower the better) as a function of the number of new nodes (denoted by m) that are added to initial graphs (each represented by the notation G_0 in the paper).
(TIF)

S4 Fig. Pairwise performance comparison between our proposed SCGG method and its competitors on the Enzymes dataset. The results are reported in terms of GED (the lower the better) as a function of the number of new nodes (denoted by m) that are added to initial graphs (each represented by the notation G_0 in the paper).
(TIF)

S5 Fig. Pairwise performance comparison between our proposed SCGG method and its competitors on the NCI1 dataset. The results are reported in terms of GED (the lower the better) as a function of the number of new nodes (denoted by m) that are added to initial graphs (each represented by the notation G_0 in the paper).
(TIF)

S6 Fig. Pairwise performance comparison between our proposed SCGG method and its competitors on the Protein dataset. The results are reported in terms of GED (the lower the better) as a function of the number of new nodes (denoted by m) that are added to initial graphs (each represented by the notation G_0 in the paper).
(TIF)

S7 Fig. Pairwise performance comparison between our proposed SCGG method and its competitors on the Protein dataset. The results are reported in terms of GED (the lower the better) as a function of the parameter m that increases discretely from 10 to 90 in steps of 10.
(TIF)

S1 File. Details regarding hyperparameter tuning.
(PDF)

S2 File. Remarks on the order of the number of graph edges.
(PDF)

S3 File. An illustrative example of the SCGG model at the evaluation time.
(PDF)

S4 File. Some potential extensions of the SCGG model.
(PDF)

Author Contributions

Conceptualization: Mahdieh Soleymani Baghshah, Hamid R. Rabiee.

Formal analysis: Faezeh Faez, Mahdieh Soleymani Baghshah, Hamid R. Rabiee.

Investigation: Faezeh Faez, Negin Hashemi Dijujin.

Methodology: Faezeh Faez, Hamid R. Rabiee.

Project administration: Hamid R. Rabiee.

Software: Faezeh Faez, Negin Hashemi Dijujin.

Supervision: Mahdieh Soleymani Baghshah, Hamid R. Rabiee.

Validation: Faezeh Faez, Negin Hashemi Dijujin, Mahdieh Soleymani Baghshah, Hamid R. Rabiee.

Visualization: Faezeh Faez.

Writing – original draft: Faezeh Faez.

Writing – review & editing: Negin Hashemi Dijujin, Mahdieh Soleymani Baghshah, Hamid R. Rabiee.

References

1. Mahmood Omar, Mansimov Elman, Bonneau Richard, and Cho Kyunghyun. Masked graph modeling for molecule generation. *Nature communications*, 12(1):1–12, 2021. https://doi.org/10.1038/s41467-021-23415-2 PMID: 34039973
2. Mahsa Ghorbani, Mojtaba Bahrami, Anees Kazi, Mahdieh Soleymani Baghshah, Hamid R Rabiee, and Nassir Navab. Gkd: Semi-supervised graph knowledge distillation for graph-independent inference. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 709–718. Springer, 2021.

3. Min Shengjie, Gao Zhan, Peng Jing, Wang Liang, Qin Ke, and Fang Bo. Stgsn—a spatial–temporal graph neural network framework for time-evolving social networks. *Knowledge-Based Systems*, 214:106746, 2021. https://doi.org/10.1016/j.knosys.2021.106746
4. Chen Ling, Tang Xing, Chen Weiqi, Qian Yuntao, Li Yansheng, and Zhang Yongjun. Dacha: A dual graph convolution based temporal knowledge graph representation learning method using historical relation. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 16(3):1–18, 2021. https://doi.org/10.1145/3161886
5. Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V Chawla. Heterogeneous graph neural network. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 793–803, 2019.
6. Huiting Hong, Hantao Guo, Yucheng Lin, Xiaoqing Yang, Zang Li, and Jieping Ye. An attention-based graph neural network for heterogeneous structural learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 4132–4139, 2020.
7. Li Jianxin, Peng Hao, Cao Yuwei, Dou Yingtong, Zhang Hekai, Yu Philip, et al. Higher-order attribute-enhancing heterogeneous graph neural networks. *IEEE Transactions on Knowledge and Data Engineering*, 2021.
8. Dexiong Chen, Leslie O'Bray, and Karsten Borgwardt. Structure-aware transformer for graph representation learning. In *International Conference on Machine Learning*, pages 3469–3489. PMLR, 2022.
9. Xiaotian Han, Zhimeng Jiang, Ninghao Liu, Qingquan Song, Jundong Li, and Xia Hu. Geometric graph representation learning via maximizing rate reduction. In *Proceedings of the ACM Web Conference* 2022, pages 1226–1237, 2022.
10. Yu Deng, Jiaolong Yang, Jianfeng Xiang, and Xin Tong. Gram: Generative radiance manifolds for 3d-aware image generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10673–10683, 2022.
11. Alisa Liu, Maarten Sap, Ximing Lu, Swabha Swayamdipta, Chandra Bhagavatula, Noah A Smith, et al. Dexperts: Decoding-time controlled text generation with experts and anti-experts. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6691–6706, 2021.
12. Chen Li-Chin, Chen Po-Hsun, Tsai Richard Tzong-Han, and Tsao Yu. Epg2s: Speech generation and speech enhancement based on electropalatography and audio signals using multimodal learning. *IEEE Signal Processing Letters*, 2022. https://doi.org/10.1109/LSP.2022.3184636
13. Li Yibo, Zhang Liangren, and Liu Zhenming. Multi-objective de novo drug design with conditional graph generative model. *Journal of cheminformatics*, 10(1):33, 2018. https://doi.org/10.1186/s13321-018-0287-6 PMID: 30043127
14. Aditya Grover, Aaron Zweig, and Stefano Ermon. Graphite: Iterative generative modeling of graphs. In *International Conference on Machine Learning*, pages 2434–2444, 2019.
15. Yang Wenju, Wen Guangqi, Cao Peng, Yang Jinzhu, and Zaiane Osmar R. Collaborative learning of graph generation, clustering and classification for brain networks diagnosis. *Computer Methods and Programs in Biomedicine*, 219:106772, 2022. https://doi.org/10.1016/j.cmpb.2022.106772 PMID: 35395591
16. Kang Minguk and Park Jaesik. Contragan: Contrastive learning for conditional image generation. *Advances in Neural Information Processing Systems*, 33:21357–21369, 2020.
17. Guo Bin, Wang Hao, Ding Yasan, Wu Wei, Hao Shaoyang, Sun Yueqi, et al. Conditional text generation for harmonious human-machine interaction. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 12(2):1–50, 2021. https://doi.org/10.1145/3439816
18. Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *International Conference on Artificial Neural Networks*, pages 412–422. Springer, 2018.
19. Yang Carl, Zhuang Peiye, Shi Wenhan, Luu Alan, and Li Pan. Conditional structure generation through graph variational generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 1340–1351, 2019.
20. Lim Jaechang, Hwang Sang-Yeon, Moon Seokhyun, Kim Seungsu, and Kim Woo Youn. Scaffold-based molecular design with a graph generative model. *Chemical Science*, 11(4):1153–1164, 2020. https://doi.org/10.1039/C9SC04503A
21. Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Hierarchical generation of molecular graphs using structural motifs. In *International Conference on Machine Learning*, 2020.
22. Yassaman Ommi, Matin Yousefabadi, Faezeh Faez, Amirmojtaba Sabour, Mahdieh Soleymani Baghshah, and Hamid R Rabiee. Ccgg: A deep autoregressive model for class-conditional graph generation. In *Companion Proceedings of the Web Conference 2022*, pages 1092–1098, 2022.

23. Zhou Tao, Lü Linyuan, and Zhang Yi-Cheng. Predicting missing links via local information. *The European Physical Journal B*, 71(4):623–630, 2009. https://doi.org/10.1140/epjb/e2009-00335-8

24. Liu Weiping and Lü Linyuan. Link prediction based on local random walk. *EPL (europhysics Letters)*, 89(5):58007, 2010. https://doi.org/10.1209/0295-5075/89/58007

25. Teji Binon, Das Jayanta K, Roy Swarup, and Bhandari Dinabandhu. Predicting missing links in gene regulatory networks using network embeddings: A qualitative assessment of selective embedding techniques. In *Intelligent Systems*, pages 143–154. Springer, 2022. https://doi.org/10.1007/978-981-19-0901-6_14

26. Erdős Paul and Rényi Alfréd. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5 (1):17–60, 1960.

27. Watts Duncan J and Strogatz Steven H. Collective dynamics of 'small-world'networks. *nature*, 393 (6684):440–442, 1998. https://doi.org/10.1038/30918 PMID: 9623998

28. Holland Paul W, Laskey Kathryn Blackmond, and Leinhardt Samuel. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983. https://doi.org/10.1016/0378-8733(83)90021-7

29. Albert Réka and Barabási Albert-László. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002. https://doi.org/10.1103/RevModPhys.74.47

30. Faez Faezeh, Ommi Yassaman, Baghshah Mahdieh Soleymani, and Rabiee Hamid R. Deep graph generators: A survey. *IEEE Access*, 9:106675–106702, 2021. https://doi.org/10.1109/ACCESS.2021.3098417

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

32. Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International Conference on Machine Learning*, pages 5708–5717, 2018.

33. Mariya Popova, Mykhailo Shvets, Junier Oliva, and Olexandr Isayev. Molecularrnn: Generating realistic molecular graphs with optimized properties. *arXiv preprint arXiv:1905.13372*, 2019.

34. Bacciu Davide, Micheli Alessio, and Podda Marco. Edge-based sequential graph generation with recurrent neural networks. *Neurocomputing*, 2020. https://doi.org/10.1016/j.neucom.2019.11.112

35. Nikhil Goyal, Harsh Vardhan Jain, and Sayan Ranu. Graphgen: A scalable approach to domain-agnostic labeled graph generation. In *Proceedings of The Web Conference 2020*, pages 1253–1263, 2020.

36. Xianduo Song, Xin Wang, Yuyuan Song, Xianglin Zuo, and Ying Wang. Hierarchical recurrent neural networks for graph generation. *Information Sciences*, 589:250–264, 2022. https://doi.org/10.1016/j.ins.2021.12.073

37. Liao Renjie, Li Yujia, Song Yang, Wang Shenlong, Hamilton Will, Duvenaud David K, et al. Efficient graph generation with graph recurrent attention networks. In *Advances in Neural Information Processing Systems*, pages 4257–4267, 2019.

38. Xiaojie Guo, Liang Zhao, Zhao Qin, Lingfei Wu, Amarda Shehu, and Yanfang Ye. Node-edge co-disentangled representation learning for attributed graph generation. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020.

39. Li Jia, Yu Jianwei, Li Jiajin, Zhang Honglei, Zhao Kangfei, Rong Yu, et al. Dirichlet graph variational autoencoder. *Advances in Neural Information Processing Systems*, 33, 2020.

40. Yuanqi Du, Yinkai Wang, Fardina Alam, Yuanjie Lu, Xiaojie Guo, Liang Zhao, et al. Deep latent-variable models for controllable molecule generation. In *2021 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 372–375. IEEE, 2021.

41. Yuanqi Du, Xiaojie Guo, Hengning Cao, Yanfang Ye, and Liang Zhao. Disentangled spatiotemporal graph generative models. In *AAAI*, 2022.

42. Yuanqi Du, Xiaojie Guo, Amarda Shehu, and Liang Zhao. Interpretable molecular graph generation via monotonic constraints. In *Proceedings of the 2022 SIAM International Conference on Data Mining (SDM)*, pages 73–81. SIAM, 2022.

43. You Jiaxuan, Liu Bowen, Ying Zhitao, Pande Vijay, and Leskovec Jure. Graph convolutional policy network for goal-directed molecular graph generation. In *Advances in neural information processing systems*, pages 6410–6421, 2018.

44. Ahn Sungsoo, Kim Junsu, Lee Hankook, and Shin Jinwoo. Guiding deep molecular optimization with genetic exploration. In *Advances in neural information processing systems*, 2020.

45. Darvariu Victor-Alexandru, Hailes Stephen, and Musolesi Mirco. Goal-directed graph construction using reinforcement learning. *Proceedings of the Royal Society A*, 477(2254):20210168, 2021. https://doi.org/10.1098/rspa.2021.0168

46. Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs. In *ICML Workshop on Theoretical Foundations and Applications of Deep Generative Models*, 2018.

47. Chence Shi, Minkai Xu, Zhaocheng Zhu, Weinan Zhang, Ming Zhang, and Jian Tang. Graphaf: a flow-based autoregressive model for molecular graph generation. In *International Conference on Learning Representations*, 2020.

48. Youzhi Luo, Keqiang Yan, and Shuiwang Ji. Graphdf: A discrete flow model for molecular graph generation. In *International Conference on Machine Learning*, pages 7192–7203. PMLR, 2021.

49. Jiao Pengfei, Guo Xuan, Jing Xin, He Dongxiao, Wu Huaming, Pan Shirui, et al. Temporal network embedding for link prediction via vae joint attention mechanism. *IEEE Transactions on Neural Networks and Learning Systems*, 2021. https://doi.org/10.1109/TNNLS.2021.3084957 PMID: 34106869

50. Ping Wang, Khushbu Agarwal, Colby Ham, Sutanay Choudhury, and Chandan K Reddy. Self-supervised learning of contextual embeddings for link prediction in heterogeneous networks. In *Proceedings of the Web Conference 2021*, pages 2946–2957, 2021.

51. Nayyeri Mojtaba, Cil Gokce Muge, Vahdati Sahar, Osborne Francesco, Rahman Mahfuzur, Angioni Simone, et al. Trans4e: Link prediction on scholarly knowledge graphs. *Neurocomputing*, 461:530–542, 2021. https://doi.org/10.1016/j.neucom.2021.02.100

52. Myunghwan Kim and Jure Leskovec. The network completion problem: Inferring missing nodes and edges in networks. In *Proceedings of the 2011 SIAM international conference on data mining*, pages 47–58. SIAM, 2011.

53. Sina Sigal, Rosenfeld Avi, and Kraus Sarit. Sami: an algorithm for solving the missing node problem using structure and attribute information. *Social Network Analysis and Mining*, 5(1):54, 2015. https://doi.org/10.1007/s13278-015-0296-7

54. Farzan Masrour, Iman Barjesteh, Rana Forsati, Abdol-Hossein Esfahanian, and Hayder Radha. Network completion with node similarity: A matrix completion approach with provable guarantees. In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*, pages 302–307. ACM, 2015.

55. Dimitrios Rafailidis and Fabio Crestani. Network completion via joint node clustering and similarity learning. In *Proceedings of the 2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 63–68. IEEE Press, 2016.

56. Tran Cong, Shin Won-Yong, Spitz Andreas, and Gertz Michael. Deepnc: Deep generative network completion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(4):1837–1852, 2022. PMID: 33074806

57. Dudziak Lukasz, Chau Thomas, Abdelfattah Mohamed, Lee Royson, Kim Hyeji, and Lane Nicholas. Brp-nas: Prediction-based nas using gcns. *Advances in Neural Information Processing Systems*, 33:10480–10490, 2020.

58. Niu Xuesong, Han Hu, Shan Shiguang, and Chen Xilin. Multi-label co-regularization for semi-supervised facial action unit recognition. *Advances in neural information processing systems*, 32, 2019.

59. Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

60. Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

61. Vaswani Ashish, Shazeer Noam, Parmar Niki, Uszkoreit Jakob, Jones Llion, Gomez Aidan N, et al. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

62. Paszke Adam, Gross Sam, Chintala Soumith, Chanan Gregory, Yang Edward, DeVito Zachary, et al. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

63. Schomburg Ida, Chang Antje, Ebeling Christian, Gremse Marion, Heldt Christian, Huhn Gregor, et al. Brenda, the enzyme database: updates and major new developments. *Nucleic acids research*, 32 (suppl_1):D431–D433, 2004. https://doi.org/10.1093/nar/gkh081 PMID: 14681450

64. Dobson Paul D and Doig Andrew J. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, 330(4):771–783, 2003. https://doi.org/10.1016/S0022-2836(03)00628-4 PMID: 12850146

65. Leskovec Jure, Chakrabarti Deepayan, Kleinberg Jon, Faloutsos Christos, and Ghahramani Zoubin. Kronecker graphs: an approach to modeling networks. *Journal of Machine Learning Research*, 11(2), 2010.

66. Himchan Park and Min-Soo Kim. Evograph: An effective and efficient graph upscaling method for preserving graph properties. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2051–2059, 2018.

67. Sanfeliu Alberto and Fu King-Sun. A distance measure between attributed relational graphs for pattern recognition. *IEEE transactions on systems*, *man*, *and cybernetics*, (3):353–362, 1983.

68. Zeng Zhiping, Tung Anthony KH, Wang Jianyong, Feng Jianhua, and Zhou Lizhu. Comparing stars: On approximating graph edit distance. *Proceedings of the VLDB Endowment*, 2(1):25–36, 2009. https://doi.org/10.14778/2428536.2428539

69. Fischer Andreas, Riesen Kaspar, and Bunke Horst. Improved quadratic time approximation of graph edit distance by combining hausdorff matching and greedy assignment. *Pattern Recognition Letters*, 87:55–62, 2017. https://doi.org/10.1016/j.patrec.2016.06.014