A Systematic Survey on Deep Generative Models for Graph Generation

Xiaojie Guo and Liang Zhao

Abstract—Graphs are important data representations for describing objects and their relationships, which appear in a wide diversity of real-world scenarios. As one of a critical problem in this area, graph generation considers learning the distributions of given graphs and generating more novel graphs. Owing to their wide range of applications, generative models for graphs, which have a rich history, however, are traditionally hand-crafted and only capable of modeling a few statistical properties of graphs. Recent advances in deep generative models for graph generation is an important step towards improving the fidelity of generated graphs and paves the way for new kinds of applications. This article provides an extensive overview of the literature in the field of deep generative models for graph generation. First, the formal definition of deep generative models for the graph generation and the preliminary knowledge are provided. Second, taxonomies of deep generative models for both unconditional and conditional graph generation are proposed respectively; the existing works of each are compared and analyzed. After that, an overview of the evaluation metrics in this specific domain is provided. Finally, the applications that deep graph generation enables are summarized and five promising future research directions are highlighted.

Index Terms—Graph generation, graph neural network, deep generative models for graphs

1 Introduction

Graphs are ubiquitous in the real world, representing objects and their relationships such as social networks, citation networks, biology networks, traffic networks, etc. Graphs are also known to have complicated structures that contain rich underlying values [1]. Tremendous efforts have been made in this area, resulting in a rich literature of related papers and methods to deal with various kinds of graph problems. These works can be categorized into two types: 1) predicting and analyzing patterns on given graphs. 2) learning the distributions of given graphs and generating more novel graphs. The first type covers many research areas including node classification, graph classification, and link prediction. Over the past few decades, a significant amount of work has been done in this domain. In contrast to the first type, the second type is related to graph generation problem, which is the focus of this paper.

Graph generation includes the process of modeling and generating real-world graphs, and it has applications in several domains, such as understanding interaction dynamics in social networks [2], [3], [4], [5], anomaly detection [6], protein structure modeling [7], [8], source code generation and translation [9], [10], and semantic parsing [11]. Owing to its many applications, the development of generative models for graphs has a rich history, resulting in famous models such as random graphs, small-world models, stochastic block models, and Bayesian network models, which generate graphs based on apriori structural assumptions [12]. These graph generation models [13], [14], [15] are engineered towards modeling a pre-selected family of graphs, such as random graphs [16], small-world networks [17], and scale-free graphs [13]. However, due to their simplicity and hand-crafted nature, these random graph models generally have limited capacity to model complex dependencies and are only capable of modeling a few statistical properties of graphs. Such methods usually fit well towards the properties that the predefined principles are tailored for, but usually cannot do well for the others. For example, a contact network models can fit flu epidemics but not dynamic functional connectivity. However, in many domains, the network properties and generation principles are largely unknown, such as those for explaining the mechanisms of mental diseases in brain network, cyber-attacks, and malware propagations. For the other example, Erdos–Rényi graphs do not have the heavy-tailed degree distribution that is typical of many real-world networks. In addition, the utilization of the apriori assumption limits these traditional techniques from exploring more applications in larger scale of domains, where the apriori knowledge of graphs are always not available.

Considering the limitations of the traditional graph generation techniques, a key open challenge is developing methods that can directly learn generative models from an observed set of graphs, which is an important step towards improving the fidelity of generated graphs. It paves the way for new kinds of applications, such as novel drug discovery [18], [19], and protein structure modeling [20], [21], [22]. Recent advances in deep generative models, such as variational autoencoders

- *Xiaojie Guo is with IBM T. J. Watson Research Center, Yorktown Height, NY 10598 USA. E-mail: xguo7@gmu.edu.*
- *Liang Zhao is with the Department of Computer Science, Emory University, Atlanta, GA 30322 USA. E-mail: liang.zhao@emory.edu.*

Manuscript received 15 August 2021; revised 31 August 2022; accepted 4 October 2022. Date of publication 17 October 2022; date of current version 3 April 2023.
This work was supported in part by NSF under Grants 2007716, 2007976, 1942594, 1907805, 1841520, and 1755850, in part by Meta Research Award, NEC Lab, Amazon Research Award, NVIDIA GPU Grant, and Design Knowledge Company under Grant 10827.002.120.04.
(Corresponding author: Liang Zhao.)
Recommended for acceptance by S. Ji.
This article has supplementary downloadable material available at https://doi.org/10.1109/TPAMI.2022.3214832, provided by the authors.
Digital Object Identifier no. 10.1109/TPAMI.2022.3214832

(VAE) [23] and generative adversarial networks (GAN) [24], have supported a number of deep learning models for generating graphs have been proposed, which formalized the promising area of *Deep Generative Models for Graph Generation*, which is the focus of this survey.

1.1 Formal Problem Definition

A graph is defined as $G(\mathcal{V}, \mathcal{E}, F, E)$, where $\mathcal{V}$ is the set of N nodes, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of M edges. $e_{i,j} \in \mathcal{E}$ is an edge connecting nodes $v_i, v_j \in \mathcal{V}$. The graph can be conveniently described in the form of matrix or tensor using its (weighed) adjacency matrix A. If the graph is node-attributed or edge-attributed, there are node attribute matrix $F \in \mathbb{R}^{N \times D}$ assigning attributes to each node or edge attribute tensor $E \in \mathbb{R}^{N \times N \times K}$ assigning attributes to each edge $e_{i,j}$. K is the dimension of the edge attributes, and D is the dimension of the node attributes.

Given a set of observed graphs $\mathbb{G} = \{G_1, \dots G_s\}$ sampled from the data distribution $p(G)$, where each graph G_i may have different numbers of nodes and edges, the goal of learning generative models for graphs is to learn the distribution of the observed set of graphs. By sampling a graph $G \sim p_{model}(G)$, new graphs can hence be achieved, which is known as deep graph generation, the short form of deep generative models for graph generation. Sometimes, the generation process can be conditioned on additional information y, such that $G \sim p_{model}(G|y)$, in order to provide extra control over the graph generation results. The generation process with such conditions is called conditional deep graph generation.

1.2 Challenges

The development of deep generative models for graphs poses unique challenges, which are mainly listed below.

Non-Unique Representations. In the general setting, a graph with n nodes can be represented by up to $n!$ equivalent adjacency matrices, each corresponding to a different, arbitrary node ordering. Given that a graph can have multiple representations, it is difficult for the models to calculate the distance between the generated graphs and ground-truth graphs white training. Thus it may require us to design either a pre-defined node ordering or a node permutation invariant reconstruction objective function.

Complex Dependencies. The nodes and edges of a graph have complex dependencies and relationships. For example, two nodes are more likely to be connected if they share common neighbors. Therefore, the generation of each node or edge cannot be modeled as an independent event. One way to formalize the graph generation is to make auto-regressive decisions, which naturally accommodate complex dependencies inside the graphs through sequential formalization of graphs. Towards this challenge, in this survey, existing works are described and compared regarding to what kinds of dependencies (e.g., dependencies among nodes, among edges or between node and edges) they can capture.

Large Output Spaces. To generate a graph with n nodes the generative model may have to output n^2 values to specify its structure, which makes it expensive, especially for large-scale graph. However, it is common to find graphs containing millions of graphs in real-world, such as citation and social networks. Consequently, it is important for generative models to scale to large-scale graphs for realistic graph generation and to accommodate such complexity in the output space. The scalability of the existing works is a critical issue in comparing and evaluating the different categories of graph generative models in this survey, as discussed in Sections 2.1.5 and 2.2.3.

Discrete Objects by Nature. The standard machine learning techniques, which were developed primarily for continuous data, do not work off-the-shelf, but usually need adjustments. A prominent example is the back-propagation algorithm, which is not directly applicable to graphs, since it works only for continuously differentiable objective functions. To this end, it is usual to project graphs (or their constituents) into a continuous space and represent them as vectors/matrix. However, reconstructing the generated graphs from the continuous representations is a challenge. Reconstructing the desecrate graph objects (i.e., nodes and edges) from continuous spaces results into different graph decoder process, such as sequentially generating the nodes of the graphs or generating the adjacent matrix of graphs in one-shot. This challenge motivates the major criteria in the taxonomy of the existing methods in this survey.

Evaluation for Implicit Properties. Evaluating the generated graphs is a very critical but challenging issue, due to the unique properties of graphs which with complex and high-dimensional structure and implicit features. Existing methods use different evaluation metrics. For example, some works [19], [25], [26] compute the distance of the graph statistic distribution of the graphs in the test set and graphs that are generated, while other works [22], [27] indirectly use some classifier-based metrics to judge whether the generated graphs are of the same distribution as the training graphs. It is important to systematically review all the existing metrics and choose the approximate ones based on their strengths and limitations according to the application requirements. Towards this challenge, we summarized a unified evaluation framework for graph generation in Section 4, including popular evaluation metrics for both unconditional and conditional graph generation.

Various Validity Requirements. Modeling and understanding graph generation via deep learning involve a wide variety of important applications, including molecule designing [18], [28], protein structure modeling [21], AMR parsing [11], [29], et al. These inter-discipline problems have their unique requirements for the validity of the generated graphs. For example, the generated molecule graphs need to have valency validity, while the semantic parsing in NLP requires Part-of-Speech (POS)-related constraint. Thus, addressing the validity requirements for different applications is crucial in enabling wider applications of deep graph generation. In this paper, we elaborate the way how the existing works improve the validity of the generated graph when introducing the rule-based generation models in Section 2.1.4. In addition, the real-world applications of solving validity issues are elaborated in Section 5.

Black-Box With Low Reliability. Compared with the traditional graph generation area, deep learning based graph modeling methods are like black-boxes which bear the weaknesses of low interpretability and reliability. Improving the interpretability of the deep graph generative models

could be a vital issue in unpacking the black-box of the generation process and paving the way for wider application domains, which are of high sensitivity and require strong reliability, such as smart health and automatic driving. In addition, semantic explanation of the latent representations can further enhance the scientific exploration of the associated application domains. Interpretability and reliability are important aspects when comparing and evaluating the different graph generation methods in this survey, as discussed in Section 3.1.3, which compares the different conditional graph generation categories.

1.3 Our Contributions

Various advanced works on deep graph generation have been conducted, ranging from the one-shot graph generation to sequential graph generation process, accommodating various deep generative learning strategies. These methods aim to solve one or several of the above challenges by works from different fields, including machine learning, bio-informatics, artificial intelligence, human health and social-network mining. However, the methods developed by different research fields tend to use different vocabularies and solve problems from different angles. Also, standard and comprehensive evaluation procedures to validate the developed deep generative models for graphs are lacking.

To this end, this paper provides a systematic review of deep generative models for graph generation. The goal is to help interdisciplinary researchers choose appropriate techniques to solve problems in their applications domains, and more importantly, to help graph generation researchers understand the basic principles as well as identify open research opportunities in deep graph generation domain. As far as we know, this is the first comprehensive survey on deep generative models for graph generation. Below, we summarize the major contributions of this survey:

- We propose a taxonomy of deep generative models for graph generation categorized by problem settings and methodologies. The drawbacks, advantages, and relations among different subcategories have been introduced.
- We provide a detailed description, analysis, and comparison of deep generative models for graph generation as well as the base deep generative models.
- We summarize and categorize the existing evaluation procedures and metrics, the benchmark datasets and the corresponding results of deep generative models for graph generation tasks.
- We introduce existing application domains of deep generative models for graphs as well as the potential benefits and opportunities they bring into these applications.
- We suggest several open problems and promising future research directions in the field of deep generative models for graph generation.

1.4 Relationship With Deep Generative Models

Deep generative models form the backbone of the base learning methods of all the existing deep generative models for graph generation. Specifically, deep generative models offer a very efficient way to analyze and understand unlabeled data. The idea behind generative models is to capture the inner probabilistic distribution that generates a class of data to generate similar data [30]. Emerging approaches such as generative adversarial networks (GANs) [24], variational auto-encoders (VAEs) [23], generative recursive neural network (generative RNN) [31] (e.g., pixelRNNs, RNN language models), flow-based learning [32], and many of their variants have led to impressive results in myriads of applications. We provide a review of five popular and classic deep generative models in Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPAMI.2022.3214832.

1.5 Relationship with Existing Surveys

There are three types of existing surveys that are relevant to our work. The first type mainly centers around the traditional graph generation by classic graph theory and network science [33], which does not focus on the most recent advancement in deep generative neural networks in AI. The second type is about representation learning on graphs [34], [35], [36], which focuses on learning graph embedding given existing graphs. Few works include a handful of deep generative models that could be used for representation learning tasks. The third type is specific to particular applications such as molecule design by deep learning, instead of for this generic technical domain.

As yet, there have been very few systematic surveys on deep generative models for graph generation, with just two recent contemporaneous papers [37], [38]. Both of these categorize graph generation mainly in terms of the general backbone learning models utilized (i.e., auto-regressive, auto-encoder-based, RL-based, adversarial, and flow-based), We have instead opted to review this research field from more comprehensive and graph-specific perspectives, including task formulation, graph generating techniques, evaluations, applications and datasets. This yields a number of advantages compared to the existing ones, namely: (1) *Two main problems are covered*: This survey comprehensively summarizes the techniques used for both unconditional and conditional generation problems; (2) *Categorization from graph-specific perspectives*: This survey categorizes the existing graph generation models (e.g., sequential-generating and one-shot generation) utilizing graph-specific perspectives, instead of the all-in purpose generative models developed and applied for all kinds of data generation; (3) *Reviews of evaluation methods*: This survey provides a comprehensive overview of the existing evaluation procedures and metrics for graph generation tasks; (4) *More applications*: This survey provides a comprehensive summary for a diverse range of the applications, including domains like biology, NLP and program analysis; and (5) *Performance comparisons*: This survey compares the performance of existing state-of-the-arts methods on both synthetic and real-world datasets, reaching several insightful conclusions.

1.6 Outline of the Survey

The remaining part of the survey is organized as follows. In Sections 2 and 3, we provide the taxonomy of deep graph generation, and the taxonomy structure is illustrated in Fig. 1. Section 2 compares related works of unconditional deep graph generation problem and summarizes the

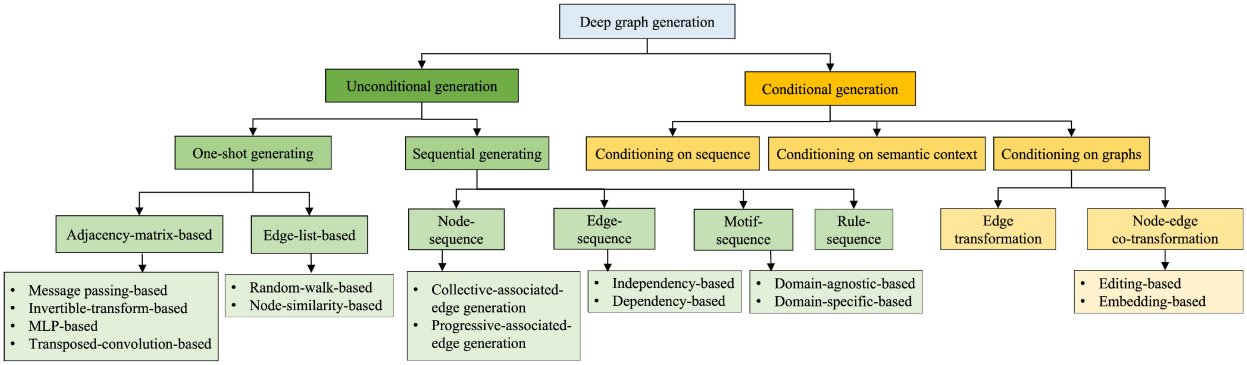


Fig. 1. Classification of deep generative models for graph generation problems.

challenges faced in each. In Section 3, we categorize the conditional deep graph generation in terms of three sub-problem settings. The challenges behind each problem are summarized, and a detailed analysis of different techniques is provided. Lastly, we summarize and categorize the evaluation metrics in Section 4. Then we present the applications that deep graph generation enables in Section 5. At last, we discuss five potential future research directions and conclude this survey in Sections 6 and 7. Due to the space limit, We also summarize the benchmark dataset and performance evaluation of existing works in Appendix B, available in the online supplemental material.

2 Unconditional Deep Generative Models for Graph Generation

The goal of unconditional deep graph generation is to learn the distribution $p_{\text{model}}(G)$ based on a set of observed realistic graphs being sampled from the real distribution $p(G)$ by deep generative models. Based on the style of the generation process, we can categorize the methods into two main branches: (1) *Sequential generating*: this generates the nodes and edges in a sequential way, one after another, (2) *One-shot generating*: this refers to building a probabilistic graph model based on the matrix representation that can generate all nodes and edges in one shot. These two ways of generating graphs have their limitations and merits. Sequential generating performs the local decisions made in the preceding one in an efficient way, but it has difficulty in preserving the long-term dependency. Thus, some global properties (e.g., scale-free property) of the graph are hard to include. Moreover, existing works on sequential generating are limited to a predefined ordering of the sequence, leaving open the role of permutation. One-shot generating methods have the capacity of modeling the global property of a graph by generating and refining the whole graph (i.e., nodes and edges) synchronously through several iterations, but most of them are hard to scale to large graphs since the time complexity is usually over $\mathbf{O}(N^2)$ because of the needs of collectively modeling global relationship among nodes. The related works following into this line of research are summarized in Table 1.

2.1 Sequential Generating

This type of methods treats graph generation as a sequential decision making process, wherein nodes and edges are generated one by one (or group by group), conditioning on the sub-graph already generated. By modeling graph generation as a sequential process, these approaches naturally accommodate complex local dependencies between generated edges and nodes. A graph G is represented by a sequence of components $S = \{s_1, \ldots, s_N\}$, where each $s_i \in S$ can be regarded as a generation unit. The distribution of graphs $p(G)$ can then be formalized as the joint (conditional) probability of all the components in general. While generating graphs, different components will be generated sequentially, by conditioning on the already generated parts.

One core issue is how to break down the graph generation to facilitate the sequential generation of its components, namely determining the formalization unit s_i for sequentialization. The most straightforward approach is to formalize the graph as a sequence of nodes, which are the basic components of a graph, to support the *node-sequence-based* generation. These methods essentially generates the graph by generating each node and its $O(N)$ associated edges in turn, and hence usually result in a total complexity of $O(N \cdot N) = O(N^2)$. Another approach is to consider a graph as set of edges, based on which a number of *edge-sequence-based* generation methods have been proposed. These methods represent the graph as a sequence of edges and generate an edge, as well as its two ending nodes, per step, which leads to a total complexity of $O(|\mathcal{E}| \cdot 2) = O(|\mathcal{E}|)$. Edge-sequence-based methods are thus usually better at sparser graphs than node-sequence-based approaches. Although both these two types are successful at retaining pairwise node-relationships, they often fall short when it comes to capture higher-order relationships [66]. For example, gene regulatory networks, neuronal networks, and social networks all contain a large number of triangles; and molecular graphs contain functional groups. These all indicate the need to generalize the units of the sequential generation from nodes/edges to interesting sub-graph patterns, known as motifs. To this end, a number of *motif-sequence-based* methods have been proposed that represent a graph utilizing a sequence of graph motifs so that a block of nodes and edges in a graph motif are generated simultaneously in each step, which usually boosts better efficiency. Although the above three types are all versatile in end-to-end graph generation, they fall short in ensuring generating "valid" graphs, namely graphs that enforce correct grammar and constraints, which are very common in fields like programming languages and molecules modeling. To solve this, several *rule-sequence-based* methods have been proposed for domain specific

TABLE 1
Deep Generative-Based Methods for Unconditional Graph Generation

Generating Style		Techniques	Reference
Sequential Generating	Node-sequence-based	Collective-associated-edge-generation	[18], [19], [39], [40], [41], [42], [43]
		Progressive-associated-edge-generation	[44], [45], [46], [47]
	Edge-sequence-based	Independency-based	[48]
		Dependency-based	[20], [49]
	Motif-sequence-based	Domain-agnostic-based	[50]
		Domain-specific-based	[28], [51], [52]
	Rule-sequence-based		[10], [53]
One-shot Generating	Adjacency-matrix-based	MLP-based	[21], [22], [54], [55], [56], [57]
		Message-Passing-based	[58], [59], [60], [61]
		Invertible-transform-based	[62], [63]
		Transposed-convolution-based	[26], [64]
	Edge-list-based	Random-walk-based	[65], [66], [67], [68]
		Node-similarity-based	[2], [69], [70], [71], [72], [73]

applications, where a graph is constructed based on a predefined sequence of rules by incorporating appropriate domain expertise. A more detailed description of methods in each category is provided below.

2.1.1 Node-Sequence-Based

General Framework. Node-sequence-based methods essentially generate the graph by generating one node and its associated edges in each step, as shown in Fig. 2a. The graph is modeled by a sequence based on a predefined ordering π on the nodes. Each unit s_i in the sequence of components S is represented as a tuple $s_i = (v_i^\pi, \{e_{i,j}\}_{j<i})$ (as shown at the bottom of Fig. 2a), indicating that at each high-level step, the generator generates one node v_i^π and all its associated edges set $\{e_{i,j}\}_{j<i}$.[1] Specifically, in node-sequence-based generation, generating a unit s_i involves two main steps. In the first step, a node is generated conditioning on the current generated graph G_i, which can be interpreted to learn $p(v_i^\pi|G_i)$. The second step is to generate the associated edges set $\{e_{i,j}\}_{j<i}$ for node v_i^π.

There are two options when it comes to generating the associated edges of each node: 1) collective associated-edge generation, where the predictions are conducted on all of the node pairs between v_i^π and the other existing nodes in G_i in a single shot to directly generate the associated edges set $\{e_{i,j}\}_{j<i}$; and 2) progressive-associated-edge generation, which generates the associated edges of node v_i^π in sequence, with two actions per step: *addEdge*, which determines the size of $\{e_{i,j}\}_{j<i}$, and *selectNode*, which determines to which node the node v_i^π will be connected if *addEdge* is needed.

Collective Associated-Edge Generation. To conduct the predictions on node pairs between the newly generated node v_i^π and all the other existing nodes, most of the works [18], [19], [39], [40], [41], [42], [70] resort to predicting the adjacent vector $A_{i,\cdot}^\pi$, which covers all the potential edges from the newly added node v_i to the other existing nodes. Thus, we can further represent each unit as $s_i = (v_i^\pi, A_{i,\cdot}^\pi)$. And the sequence can be represented as $Seq(G, \pi) = \{(v_1^\pi, A_{1,\cdot}^\pi), \ldots, (v_N^\pi, A_{N,\cdot}^\pi)\}$. The aim is to learn the distribution as

$$p(\mathcal{V}^\pi, A^\pi) = \prod_{i=1}^{N} p(v_i^\pi | v_{<i}^\pi, A_{<i,\cdot}^\pi) p(A_{i,\cdot}^\pi | v_{\leq i}^\pi, A_{<i,\cdot}^\pi), \quad (1)$$

where $v_{<i}^\pi$ refers to the nodes generated before v_i^π and $A_{<i,\cdot}^\pi$ refers to the adjacent vectors generated before $A_{i,\cdot}^\pi$. Such joint probability can be implemented by sequential-based architectures such as generative RNN models [18], [19], [27], [41] and auto-regressive flow-based learning models [70]. Here we introduce the RNN-based models as an example.

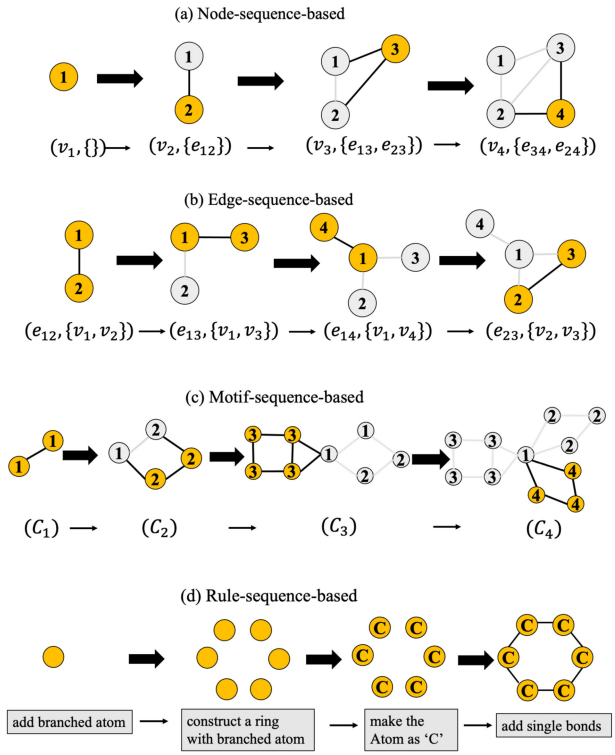


Fig. 2. Four categories in graph sequential generating: the upper line of each sub-figure refers to the immediate graph that are generated per step; the bottom line of each sub-figure refers to the sequence consisting of unit S_i that is generated per step.

1. Here we omit the node and edge attribute symbol for clarity, but it is important to bear in mind that the generated node and edges can all have attributes (i.e., type, label).

In the generative RNN-based models, the node distributions $p(v_i^\pi|v_{<i}^\pi, A_{<i,\cdot}^\pi)$ are typically assumed as a multivariate Bernoulli distribution that is parameterized by $\phi_i \in \mathbb{R}^T$, where T refers to the number of node categories. The edge existence distribution $p(A_{i,\cdot}^\pi|v_{\leq i}^\pi, A_{<i,\cdot}^\pi)$ can be assumed as the joint probability of several dependent Bernoulli distributions

$$p(A_{i,\cdot}^\pi|A_{<i,\cdot}^\pi) = \prod_{j=1}^{i-1} p(A_{i,j}^\pi|A_{i,<j}^\pi, A_{<i,\cdot}^\pi), \tag{2}$$

where $p(A_{i,\cdot}^\pi|A_{<i,\cdot}^\pi)$ is parameterized by $\theta_i \in \mathbb{R}^{i-1}$ and the distribution of $p(A_{i,j}^\pi|A_{i,<j}^\pi, A_{<i,\cdot}^\pi)$ is parameterized by each entry $\theta_{i,j}$ in θ_i. The architecture for implementing Eqs. (1) and (2) can be regarded as a hierarchical-RNN, where the outer RNN is used for generating the nodes and the inner RNN is used for generating each node's associated edges. After either a node or edge is generated, a graph-level hidden representation of the already generated sub- graph is updated through a message passing neural network (MPNN) [74]. Specifically, at each Step i, a parameter ϕ_i will be calculated through a multilayer perceptron (MLP)-based function based on the current graph-level hidden representation. The parameter ϕ_i is used to parameterize the Bernoulli distribution of node existence, from which node v_i^π is sampled. After that, the adjacent vector $A_{i,\cdot}^\pi$ is generated by sequentially generating each of its entry.

Progressive Associated-Edge Generation. The above-introduced collective associated-edge generation has a time complexity of $O(N^2)$ that is time-consuming especially for sparse graphs. A remedy is to progressively select the nodes to be connected with the current node v_i^π from the existing nodes $v_{<i}^\pi$, until the desired number of nodes is selected, which is small for sparse graph. Specifically, for the current node v_i^π, we generate $\{e_{i,j}\}_{j<i}$ by applying two functions: 1) an *addEdge* function to determine the size of the edge set $\{e_{i,j}\}_{j<i}$ of node v_i^π and 2) a *selectNode* function to select the nodes to be connected from the existing graph [44], [45], [46], [47]. The complexity of progressive associated-edge generation method is $O(MN)$ where M refers to the number of edges.

Specifically, after generating a node v_i^π in the first step, an *addEdge* function is used to output a parameter as $f_{addEdge}(h_{v_i}^\pi)$, following a Bernoulli distribution indicating whether to add an edge to the node v_i^π. Here $h_{v_i}^\pi$ refers to the node-level hidden states of v_i^π which is calculated through a node embedding function, e.g., MPNN [74] based on the already-generated parts of the graph. If an edge is determined to be added, the next step is selecting the neighboring node v_j^π from the existing nodes. To achieve this, we can compute a score m_j^π (as Eq. (3)) for each existing node v_j^π based on *selectNode* function $f_{selectNode}$, which is then passed through a softmax function [75] to be properly normalized into a distribution of nodes

$$m_{i,j}^\pi = f_{selectNode}(h_{v_i}^\pi, h_{v_j}^\pi), \quad where j < i. \tag{3}$$

$$p(e_{i,j}|v_{<i}^\pi, \{e_{<i,j}\}_{j<i}) = softmax(m_{i,j}^\pi). \tag{4}$$

The MLP-based function $f_{selectNode}$ maps pairs of node-level hidden states $h_{v_i}^\pi$ and $h_{v_j}^\pi$ to a score $m_{i,j}^\pi$ for connecting node v_j^π to the new node v_i^π. This can be extended to handle discrete edge attributes by making $m_{i,j}^\pi$ a vector of scores with the same size as the number of the edge attribute's categories, and taking the softmax over all categories of the edge attribute. Based on the aforementioned procedure, the two functions $f_{addEdge}$ and $f_{selectNode}$ are iteratively executed to generate the edges within the edge set $\{e_{<i,j}\}_{j<i}$ of node v_i^π until the terminal signal from function $f_{addEdge}$ indicates that no more edges for node v_i are yet to be added.

2.1.2 Edge-Sequence-Based

General Framework. Edge-sequence-based methods [20], [48], [49] consider a graph to be a sequence of edges and generate an edge, along with its two end nodes, in each step, as shown in Fig. 2b. It defines an ordering of the edges in the graph and also an ordering function $\alpha(\cdot)$ for indexing the nodes. The graph G can then be modeled by a sequence of edges, with each unit in the sequence being a tuple represented by $s_i = (\alpha(u), \alpha(v), F_u, F_v, E_{u,v}^i)$, where each element of the sequence consists of a pair of nodes' indexes $\alpha(u)$ and $\alpha(v)$ for nodes u and v, node attribute F_u, F_v, and the edge attribute $E_{u,v}^{(i)}$ for the edge at Step i. In edge-sequence-based generation, there are two ways to generate a unit s_i, with the first based on the assumption that $\alpha(u)$ and $\alpha(v)$ are mutually independent while the second assumes they are mutually dependent, with their details as follows.

Independency-Based. Goyal et al. [48] used depth first search (DFS) algorithm [76] as the ordering index function $\alpha(\cdot)$ to construct graph canonical index of nodes. The conditional distribution for generating each edge in graph G can be formalized as

$$\begin{aligned} p(s_i|s_{<i}) &= p((\alpha(u), \alpha(v), F_u, F_v, E_{u,v}^i)|s_{<i}) \\ &= p(\alpha(u)|s_{<i})p(\alpha(v)|s_{<i})p(F_u|s_{<i})p(F_v|s_{<i})p(E_{u,v}^i|s_{<i}), \end{aligned} \tag{5}$$

where $s_{<i}$ refers to the already-generated edges and nodes. A customized long short-term memory (LSTM) is designed which consists of a transition state function f_{trans} for transferring the hidden state of the last step into that of the current step (in Eq. (6)), an embedding function f_{emb} for embedding the already generated graph into latent representations (in Eq. (6)), and five separate output functions for the above five distribution components (in Eqs. (6), (7), (8), (9), (10), and (11)). It is assumed that the five elements in one tuple are independent of each others, and thus the inference is as

$$h_G^{(i)} = f_{\text{trans}}(h_G^{(i-1)}, f_{\text{emb}}(s_{i-1})) \tag{6}$$

$$\alpha(u) \sim Cat(\theta_{\alpha(u)}); \quad \theta_{\alpha(u)} = f_{\alpha(u)}(h_G^{(i)}); \tag{7}$$

$$\alpha(v) \sim Cat(\theta_{\alpha(v)}); \quad \theta_{\alpha(v)} = f_{\alpha(v)}(h_G^{(i)}) \tag{8}$$

$$F_u \sim Cat(\theta_{F_u}); \quad \theta_{F_u} = f_{F_u}(h_G^{(i)}); \tag{9}$$

$$F_v \sim Cat(\theta_{F_v}); \quad \theta_{F_v} = f_{F_v}(h_G^{(i)}) \tag{10}$$

$$E_{u,v}^i \sim Cat(\theta_{E_{u,v}^i}); \quad \theta_{E_{u,v}^i} = f_{E_{u,v}^i}(h_G^{(i)}), \tag{11}$$

where s_{i-1} refers to the generated tuple at Step $i-1$ and is represented as the concatenation of all the component representations in the tuple. $h_G^{(i)}$ is a graph-level LSTM hidden state vector that encodes the state of the graph generated so far at Step i. Given the graph state $h_G^{(i)}$, the output of five

functions $f_{\alpha(u)}$, $f_{\alpha(v)}$, f_{F_u}, f_{F_v}, $f_{E_{u,v}}$ model the categorical distribution of the five components of the newly formed edge tuple, which are paramerized by five vectors $\theta_{\alpha(u)}$, $\theta_{\alpha(v)}$, θ_{F_u}, θ_{F_v}, $\theta_{E_{u,v}}$ respectively. Finally, the components of the newly formed edge tuple are sampled from the five learnt categorical distributions.

Dependency-Based. To further characterize the dependency between $\alpha(u)$ and $\alpha(v)$, Bacciu et al. [20] assume the existence of node dependence in a tuple. This method deals with homogeneous graphs without considering the node/edge categories, by representing each tuple in the sequence as $s_i = (\alpha(u), \alpha(v))$ and formalizing the distribution as $p(s_i|s_{<i}) = p(\alpha(u)|s_{<i})p(\alpha(v)|\alpha(u), s_{<i})$. Then, the first node is sampled in the same way as in Eq. (8), while the second node in the tuple is sampled as follows:

$$\alpha(v) \sim Cat(\theta_{\alpha(v)}); \quad \theta_{\alpha(v)} = f_{\alpha(v)}(h_G^{(i)}, g_{\text{emb}}(\alpha(u))), \tag{12}$$

where the function g_{emb} is used for embedding the index of the first generated node u in the pair.

2.1.3 Motif-Sequence-Based

General Framework. Motif-sequence-based methods [28], [50], [51], [52] represent a graph G as a sequence of graph motifs, $Seq(G) = \{C_1, \ldots, C_M\}$, where the block of nodes and edges that constitute each graph motif C_i are generated at each step, as shown in Fig. 2c. A new graph-motif C_i is generated in each step by conditioning on the current graph G_i at Step i and then it is connected to G_i.

A key problem in motif-based methods is how to connect the newly generated graph motif C_i to G_i, given that there are many potential ways to link two sub-graphs. These linking strategies are highly dependent on the definition of the graph motifs. For *Domain-agnostic* graphs, given a predefined node ordering, the graph motifs are usually defined as a combination of consecutive nodes. This allows us to predict the associated edges of all the nodes in C_i and connect it to G_i based on these predictions. For *Domain-specific* graphs, the motifs are usually defined and connected based on specific domain knowledge, such as chemical motifs for a task involving molecular structure generation.

Domain-Agnostic-Based. This line of works is designed for generating general graphs without the need of domain expertise; it is similar to the collective-associated-edge-generation category under the line of node-sequence generation by generating the adjacent vectors for each edge, such as GraphRNN [19], except for the generation of several nodes instead of one per step. As described in Section 2.1.1, a graph G is represented as a sequence of node-based tuples as $G = \{s_1, \ldots, s_N\}$, where $s_i = (v_i^\pi, A_{i,:}^\pi)$ is generated per step. Based on this node sequence, Liao et al. [50](GRANs) regard every tuples consisting of B recursive nodes as a graph motif C_i and generates each block per step. In this way, the generated nodes in the new graph motif follow the ordering of the nodes in the whole graph and contain all the connection information of the existing and newly generated nodes. To formalize the dependency among the existing and newly generated nodes, GRANs proposes an MPNN-based model to generate the adjacent vectors. Specifically, for the ith generation step, a graph G_i is generated which contains the already-generated graph with $B \cdot (i-1)$ nodes and the edges among them, as well as the B nodes in the newly generated graph motif. For these new B nodes, edges are initially completely added to connect all of them with each other and the previous $B \cdot (i-1)$ nodes. Then an MPNN-based graph neural network (GNN) [77] on this augmented graph is used to update the nodes' hidden states by encoding the graph structure. After several rounds of message passing implementation, the node-level hidden states of both the existing and newly added nodes are used to infer the final distribution of the newly added edges as

$$p(C_t|C_{<t}) = \prod_{B(t-1)<i\leq B} \prod_{1\leq j\leq i} p(A_{i,j}^\pi|C_{<t}), \tag{13}$$

where the Bernoulli distribution $p(A_{i,j}^\pi|C_{<t})$ is parameterized for modeling the edge existence through an MLP, which takes the node-level hidden states as input.

Domain-Specific-Based. The definition of graph motifs and its connections can involve domain knowledge, such as in the situation of molecules generation (i.e., graph of atoms) [28], [51]. Jin et al. [28] propose the Junction-Tree-VAE by first generating a tree-structured scaffold over chemical substructures, and then combining them into a molecule with an MPNN. Specifically, a Tree Decomposition of Molecules algorithm [78] tailored for molecules to decompose the graph G into several graph motifs C_i is followed, and each C_i is regarded as a node in the tree structure. The other way of defining the graph motifs is to leverage the breaking of retrosynthetically interesting chemical substructures (BRICS) algorithm [79]. To generate a graph G, a tree is first generated and then converted into the final graph. The decoder for generating a T consists of both *topology prediction function* and *label prediction function*. The topology prediction function models the probability of the current node to have a child, and the label prediction function models a distribution of the labels of all types of C_i. When reproducing a molecular graph G that underlies the predicted junction tree T, since each motif contains several atoms, the neighboring motifs C_i and C_j can be attached to each other as sub-graphs in many potential ways. To solve this, a scoring function (e.g., measuring the validness of the potentially generated graph) over all the candidates graphs is proposed, and the optimal one that maximizes the scoring function is the final generated graph.

2.1.4 Rule-Sequence-Based

General Framework. Several methods that have been proposed [10], [53] generate a sequence of production rules or commands to guide the graph construction process sequentially. This is usually the method of choice where the targeted graph has strong constraints or grammar that must be satisfied in order to construct a valid graph. For example, a molecule can not violate fundamental properties like charge conservation, which thus constrains the patterns available for the node types and edges of molecule graph. To ensure the validity of the generated graphs, graph generation is transformed by generating parse trees, which describe a discrete molecular structure utilizing context free grammar (CFG), while the parse tree itself can be further expressed as a sequence of rules based on a pre-defined order.

Kusner et al. [53] propose generating a parse tree that describes a discrete object (e.g., arithmetic expressions and molecule) by a grammar; they also proposed a graph generation method named GrammerVAE. An example of using the parse tree for molecule generation: to encode the parse tree, they decompose it into a sequence of production rules by performing a pre-ordered traversal on its branches from left-to-right, and then convert these rules into one-hot indicator vectors, where each dimension corresponds to a rule in the SMILES grammar. The deep convolutional neural network is then mapped into a continuous latent vector z. While decoding, the continuous vector z is passed through an RNN which produces a set of unnormalized log probability vectors (i.e., "logits"). Each dimension of the logit vectors corresponds to a production rule in the grammar. The model generates the parse trees directly in a top-down direction, by repeatedly expanding the tree with its production rules. The molecules are also generated by following the rules generated sequentially, as shown in Fig. 2d. Although the CFG provides a mechanism for generating syntactic-valid objects, it is still incapable of guaranteeing the model for generating semantic valid objects [53]. To deal with this limitation, Dai et al. [10] propose the syntax-directed variational autoencoder (SD-VAE), in which a semantic restriction component is advanced to the stage of syntax-tree generator. This allows for a the generator with both syntactic and semantic validity.

2.1.5 Comparison of Different Sub-Categories

In this subsection, we compare the four categories of sequential-generating method from three aspects: (1) *Scalability*: time complexity determines the scalability of the graph generation methods. Node-sequence-based methods commonly have the time complexity of $O(N^2)$ when N denotes to the number of nodes, while edge-sequence-based methods usually have the complexity of $O(|\mathcal{E}|)$. Thus, for sparse graphs where $N^2 \gg |\mathcal{E}|$, edge-sequence-based methods are more scalable than node-sequence-based ones. The complexities of motif-sequence-based methods vary from $O(N^2)$ (e.g., for domain-agnostic type) to $O(N \cdot |C|)$ (e.g., for domain-specific type), where $|C|$ refers to the number of motifs. The complexity of rule-sequence-based methods usually linearly related to the number of rules in generating a graph; (2) *Expressiveness*: the expressiveness of generation model relies on its power to model the complex dependency among the objects in the graph. Node-sequence and edge-sequence generation can capture the most sophisticated dependence, including node-node dependence, edge-edge dependence and node-edge dependence. While the motif-sequence-based methods are able to model the dependence between graph-motifs which capture the high-order relationships and global patterns. Rule-sequence-based methods can model the dependency between the operation rules to capture the semantic patterns in building a realistic graphs, which are usually difficult to directly learn from the graph topology; (3) *Application scenarios*: the selection of categories of sequential generating techniques for a specific application scenario depends on its sensitivity to validness and the accessibility of the generation rules. Node- and edge-sequence-based methods are suitable in generating realistic graphs without the domain expertise (e.g., the known rules, constrains or candidate motifs), such as the social and traffic networks. Motif-sequence-based methods can partially guarantee the validness of the generated graph by selecting graph-motifs from the predefined valid motif candidates. Rule-sequence-based methods are more powerful in generating valid realistic graphs by following the correct grammar and constraints. Thus, the latter two types of methods are preferred in validness-sensitive applications, such as molecule generation and program modeling.

2.2 One-Shot Generating

One-shot generating methods learn to map each whole graph into a unified latent representation which follows some probabilistic distribution in latent space. Each whole graph can then be generated by directly sampling from this probabilistic distribution in one step. The core issue of these methods is usually how to jointly generate graph topology together with node/edge attributes. Considering that the graph topology can usually be represented in terms of adjacency matrix and node attribute matrix, the typical solution is to learn the distribution of these two and generate them in one shot, which is categorized as *Adjacent-matrix-based* generation. Learning the distribution of adjacency matrices is potentially expressive yet comes with inefficiency issue in both memory and time. To this end, *Edge-list-based* methods learn the local patterns and hence is usually good at handling larger graphs with simpler global patterns.

2.2.1 Adjacency-Matrix-Based

General Framework. Adjacency-matrix-based methods build models to directly map the latent embedding z to the output graph in terms of an adjacency matrix, generally with the addition of node/edge attribute matrices/tensors. Hence, how to best achieve an expressive and efficient mapping is the core challenge and there is usually a trade-off between them. Existing techniques are built upon popular deep neural network scenarios that are MLP-based, message-passing-based, invertible-transformation-based or transposed-convolution-based. MLP-based models are highly end-to-end, while message-passing-based approaches and transposed-convolution-based can explicitly model higher-order correlations in graphs. Invertible-transformation-based techniques more rigorously model invertible mappings but impose more limitations on the expressiveness.

MLP-Based Methods. Most of the one-shot graph generation techniques involves simply constructing the graph decoder $g(z)$ using MLP [21], [22], [54], [55], [56], [57], where the models' parameters can be optimized under common frameworks such as VAE and GAN. The MLP-based models ingest a latent graph representation $z \sim p(z)$ and simultaneously output adjacent matrix A^π and node attribute F^π, as shown in Fig. 3a. Specifically, the generator $g(z)$ takes D-dimensional vectors $z \in \mathbb{R}^D$ sampled from a statistical distribution such as standard normal distribution and outputs graphs. For each z, $g(z)$ outputs two continuous and dense objects: $\tilde{A}^\pi$, which defines edge attributes and $\tilde{F}^\pi$, which denotes node attributes through two simple MLPs. Both $\tilde{A}^\pi$ and $\tilde{F}^\pi$ have a probabilistic interpretation since each node and edge attribute is represented with probabilities of

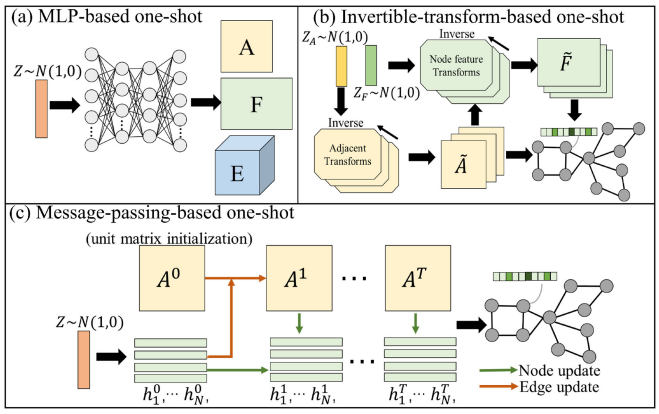


Fig. 3. Schema for adjacent-matrix-based one-shot generation.

categorical distributions of types. To generate the final graph, it is required to obtain the discrete-valued objects A^{π} and F^{π} from $\tilde{A}^{\pi}$ and $\tilde{F}^{\pi}$, respectively. The existing works have two ways to realize this step detailed as follows.

In the first way, the existing works [21], [54], [55] use sigmoid activation function to compute A^{π} and F^{π} during the training time. At test time, the discrete-valued estimate A^{π} and F^{π} can be obtained by taking edge- and node-wise argmax in $\tilde{A}^{\pi}$ and $\tilde{F}^{\pi}$. Alternatively, existing works [22], [56], [57] leverage categorical reparameterization with the Gumbel-Softmax [80], [81], which is to sample from a categorical distribution during the forward pass (i.e., $F_i^{\pi} \sim Cat(\tilde{F}_i^{\pi})$ and $A_{ij}^{\pi} = Cat(\tilde{A}_{ij}^{\pi})$) and the original continuous-valued $\tilde{A}^{\pi}$ and $\tilde{F}^{\pi}$ in the backward pass. In this way, these methods can perform continuous-valued operations during the training procedure and do the categorical sampling procedure to finally generate F and A.

Message-Passing-Based Methods. Message-passing-based methods generate graphs by iteratively refining the graph topology and node representations of the initialized graph through the MPNN. Specifically, based on the latent representation z sampled from a simple distribution (e.g., Gaussian), we usually first generate an initialized adjacent matrix A^0 and the initialized node latent representations $H^0 \in \mathbb{R}^{N \times L}$, where L refers to the length of each node representation (here we omit the node ordering symbol π for clarity). Then A^0 and H^0 are updated though MPNN into A^1 and H^1, which are the adjacent matrix and hidden states in the first intermediate layer, then another MPNN layer is applied to generate for the 2nd layer, etc. We can stack multiple such layers to explicitly characterize the higher-order correlation among nodes and edges. Each MPNN layer can be expressed as follows:

$$A_{i,j}^{l+1} = A_{i,j}^{l} + ReLu(\nu_1 A_{i,j}^{l} + \nu_2 h_i^{l} + \nu_3 h_j^{l});$$
$$h_i^{l+1} = h_i^{l} + ReLu(w_1\ h_i^{l} + \sum\nolimits_j^{N} \eta_{i,j} w_2\ h_j^{l}), \quad (14)$$

where ν_1, ν_2, ν_3, w_1 and w_2 are trainable parameters. We can stack multiple such layers to explicitly characterize the higher-order correlation among nodes and edges, which is also illustrated in Fig. 3c. Finally, after T layers' updating, the outputs $A_{i,j}^{T}$ and F_i^{T} are used to parameterize the categorical distributions of each edge and node, based on which each edge $A_{i,j}$ and node F_i are generated through categorical sampling introduced above. To learn the above generator, Existing methods leverage various learning frameworks such as VAE and GANs [58], [59], [60], or have a plain framework based on the score-based generation [61].

Invertible-Transform-Based Methods. Flow-based generative methods can also do one-shot generation, by a unique invertible function between graph G and the latent prior z sampling from a simple distribution (e.g., Gaussian), as shown in Fig. 3b. Concretely, based on vanilla flow-based learning techniques introduced in Appendix. Special forward transformation $G \underrightarrow{\ } z$ and backward transformation $z \underrightarrow{\ } G$ needs to be designed.

Madhawa et al. [63] propose the first flow-based one-shot graph generation model called GraphNVP. To get $z = (z^F, z^A)$ from $G = (A, F)$ in the *forward transformation*, they first convert the discrete variable A and F into continuous variable A' and F' by adding real-valued noise), which is known as *dequantization*. Then two types of reversible affine coupling layers: adjacency coupling layers and node attribute coupling layers are utilized to transform the adjacency matrix A' and the node attribute matrix F' into latent representations z_A and z_F, respectively. The lth reversible coupling layers are designed as follows:

$$z_F^{l}[i] = z_F^{l-1}[i] \odot \exp(s_F(z_F^{l-1}[i], A)) + t_F(z_F^{l-1}[i], A) \quad (15)$$
$$z_A^{l}[i,j] = z_A^{l-1}[i,j] \odot \exp(s_A(z_A^{l-1}[i,j])) + t_A(z_A^{l-1}[i,j]), \quad (16)$$

where $z_F^0 = X'$ and $z_A^0 = A'$. $z_F^l[i]$ refers to the ith entry of z_F^l; $\odot$ denotes element-wise multiplication. Functions $s_A(\cdot)$ and $t_A(\cdot)$ stand for scale and translation operations which can be implemented based on MPNN, and $s_F(\cdot)$, $t_F(\cdot)$ can be implemented based on MLP networks. To get $G = (F, A)$ from $z = (z_F, z_A)$ in the *backward transformation*, the reversed operation is conducted based on the above forward transformation operation in Eqs. (15) and (16). Next a probabilistic feature matrix $\tilde{F}$ is generated given the sampled z_F and the generated adjacency matrix A through a sequence of inverted node attribute coupling layers. Likewise, the node-wise argmax of $\tilde{F}$ is used to get discrete feature matrix F.

Transposed-Convolution-Based Methods. One typical type of graph decoder in the one-shot-generation techniques is constructed based on the transposed convolution neural networks [26]. The process is about generating the adjacent matrix of graph by taking the node latent representation vectors as input. The transposed-convolution-based decoder consists of a node transposed convolution layer and several edge transposed convolution layers.

The node transposed convolution layer is used to decode the edge representations of the graph based on the node embedding. For example, after a node transposed convolution layer, the edge representations $E_{i,j}$ between node v_i and node v_j can be computed as

$$E_{i,j} = \sum\nolimits_{m=1}^{L} (\sigma(H_i^m \bar{\mu}_j) + \sigma(H_j^m \bar{\nu}_i)), \quad (17)$$

where $\sigma(H_i^m \bar{\mu}_j)$ means the transposed convolution contribution of node v_i to its potential edge $E_{i,j}$, which is made by the mth entry of its node representations, and $\bar{\mu}_j$ represents one entry of the transposed convolution filter vector $\bar{\mu} \in$

$\mathbb{R}^{N\times 1}$ that is related to node v_j. L refers to the length of the node representation.

Several edge transposed convolution layers are recursively applied to decode the latent edge representations from the upper layer back to those of the lower layer. Thus, $E^l_{i,j}$ between node v_i and node v_j in the $(l+1)$th layer is computed as

$$E^{l+1}_{i,j} = \sigma(\bar{\phi}^l_j \sum\nolimits_{k_1=1}^{N} E^l_{i,k_1} S_{k_1}) + \sigma(\bar{\psi}^l_i \sum\nolimits_{k_2=1}^{N} E^l_{k_2,j} S_{k_1}), \quad (18)$$

where $\bar{\phi}^l_j \sum_{k_1=1}^{N} E^l_{i,k_1} S_{k_1}$ can be interpreted as the decoded contribution of node v_i to its related edge representations $E^{l+1}_{i,j}$, and $\bar{\phi}^l_j$ refers to the element of transposed convolution filter vector that is related to node v_j. σ refers to the activation functions.

2.2.2 Edge-List-Based

General Framework. This category typically requires a generative model that learns edge probabilities, where all the edges are generated independently. These methods are usually applied when learning from one large-scale graph to generate a new one using the existing nodes. The general pipeline is composed of two main steps. A score is calculated for each edge (i.e., pair of nodes) to estimate the edge probability, after which the edges can be sampled.

In terms of how the edge probabilities are generated, existing works are further categorized as either *random-walk-based* [65], [66], [67], [68] or *node-similarity-based* [2], [69], [71], [72], [73]. *Node-similarity-based* models calculate the edge probability based on the similarity of each pair of node representations learnt from graphs, while *random-walk-based* methods estimate each edge probability by calculating the edge frequency for a large set of random walks generated by sampling from their distributions learnt from graphs.

Random-Walk-Based. This type of methods generate the edge probability based on a score matrix, which is calculated by the frequency of each edge that appears in a set of generated random walks. NetGAN [65] is proposed to mimic the large-scale real-world networks. Specifically, at the first step, a GAN-based generative model is used to learn the distribution of random walks over the observed graph, and then it generates a set of random walks. At the second step, a score matrix $S \in \mathcal{R}^{N\times N}$ is constructed, where each entry denotes the counts of an edge that appears in the set of generated random walks. At last, based on the score matrix, the edge probability matrix $\tilde{A}$ is calculated as $\tilde{A}_{i,j} = S_{i,j/} \sum_{u,v}^{N} S_{u,v}$, which will be used to generate individual edge $A_{i,j}$, based on efficient sampling processes.

Following this, some works propose improving the NetGAN, by changing the way to choose the first node in starting a random walk [68] or learning spatial-temporal random walks for spatial-temporal graph generation [67]. Gamage et al. [66] generalize the NetGAN by adding two motif-biased random-walk GANs. The edge probability is thus calculated based on the score matrices from three sets of random walks (i.e., $S^{(1)}$, $S^{(2)}$, and $S^{(3)}$) that are generated from the three GANs. To sample each edge, one view $S^{(k)}$ is randomly selected from the three scores matrices. Based on $S^{(k)}$, edge probability $\tilde{A}_{i,j}$ is calculated as $\tilde{A}_{i,j} = S^{(k)}_{i,j} / \sum_{u,v}^{N} S_{u,v}$.

Node-Similarity-Based. These methods generate the edge probability based on pairwise relationships between the given or sampled nodes' embedding (as in [69]). Specifically, the probability adjacent matrix $\tilde{A}$ is generated given the node representations $Z \in \mathcal{R}^{N\times L}$, where $Z_i \in \mathcal{R}^L$ refers to the latent representation for node v_i. $\tilde{A}$ will be used to generate individual edge $A_{i,j}$, based on efficient sampling processes. Existing methods differ on how to calculate $\tilde{A}$.

Several works [2], [69], [71] compute $\tilde{A}_{i,j}$ based on the inner-product operations of two node embedding Z_i and Z_j. This reflects the idea that nodes that are close in the embedding space should have a high probability of being connected. These works require a setting where node set is pre-defined and the node attribute F is known in advance. Specifically, by first sampling node latent representation Z_i from the standard normal distribution, Kipf et al. [2], [69] calculate the probability adjacent matrix as $\tilde{A} = \mathrm{Sigmoid}(ZZ^T)$. The adjacent matrix A is then sampled from $\tilde{A}$ which parameterizes the Bernoulli distribution of the edge existence, as similar to work in [71]. Other works [72], [73] compute $\tilde{A}_{i,j}$ by measuring the closeness of two node, representations with ℓ_2 norm. Liu et al. [72] propose a decoder for calculating $\tilde{A}_{i,j}$ as

$$\tilde{A}_{i,j} = 1/(1 + exp(C(\parallel Z_i - Z_j \parallel_2^2 - 1))), \quad (19)$$

where C is called a temperature hyperparameter. Salha et al. [73] propose a gravity-inspired decoding schema as

$$\tilde{A}_{i,j} = \mathrm{Sigmoid}(m_j - log \parallel Z_i - Z_j \parallel_2^2), \quad (20)$$

where m_j is the gravity scale of node v_j learned from the input graph by its featured encoder.

2.2.3 Comparison of Different Sub-Categories

In this subsection, we compare two aspects of the two different types of one-shot method: (1) *Time complexity*: Both adjacent-matrix-based and node-similarity-based edge-list generation have a complexity of $O(N^2)$ since they need to consider every pairs of N nodes in the graph. Random-walk-based edge-list generation is more scalable, as here the edges are sampled based on the edge probability, which is determined by the edge frequency in a set of generated random walks; and (2) *Application scenarios*: Since adjacent-matrix-based methods can handle global patterns with high expressiveness and minimum time consumption, these types of methods are widely used for small graphs (i.e., graphs with less than 1,000 nodes) whose global patterns are important, such as molecules and proteins. Edge-list-based methods, on the other hand, are efficient when it comes to generating large graphs whose local patterns are important, such as social networks and citation networks.

3 Conditional Deep Generative Models for Graph Generation

The goal of conditional deep graph generation is to learn a conditional distribution $p_{\mathrm{model}}(G|y)$ based on a set of observed realistic graphs G along with their corresponding auxiliary information, namely a condition y. The auxiliary

TABLE 2
Deep Generative-Based Methods for Conditional Graph Generation

Conditioning objects		Techniques of encoding conditions	References
Graphs	Edge Transformation	Adjacent-based edge convolution	[26], [64], [82], [83]
	Node-edge Co-transformation	Embedding-based	[25], [84], [85], [86]
		Editing-based	[87], [88], [89]
Sequence		RNN-based encoding	[90], [91], [92], [93]
Context Semantics		Concatenation with latent representation	[45], [94], [95], [96]

information could be category labels, semantic context, graphs from other distribution spaces, etc.

Compared with unconditional deep graph generation, in addition to the challenge in generating graphs, conditional generation needs to consider how to extract the features from the given condition and integrate them into the generation of graphs. Thus, to systematically introduce the existing conditional deep graph generative models, we mainly focus on describing how these methods deal with the conditions. Since the conditions could be any form of auxiliary information, they are categorized into three types, including *graphs*, *sequence*, and *semantic context*, shown as the yellow parts of the taxonomy tree in Fig. 1.

3.1 Conditioning on Graphs

The problem of deep graph generation conditioning on another graph is also called as deep graph transformation (also known as deep graph translation) [26]. It aims at translating an input graph G_S in the source domain to the corresponding output graphs G_T in the target domain. Considering the entities that are being transformed during the translation process, there are two categories of works in the domain of deep graph generation conditioning on graphs: *edge transformation* and *node-edge-co-transformation*, as shown in Table 2.[2]

3.1.1 Edge Transformation

Overall Problem Formulation. The problem of edge transformation is to generate the graph topology and edge attributes of the target graph conditioning on the input graph. It requires the edge set $\mathcal{E}$ and edge attributes E to change while the graph node set and node attributes are fixed during the translation process as: $\mathcal{T} : G_S(\mathcal{V}, \mathcal{E}_S, F, E_S) \rightarrow G_T(\mathcal{V}, \mathcal{E}_T, F, E_T)$. The edge transformation problem has a wide range of real-world applications, such as modeling chemical reactions [87], protein folding [21] and malware cyber-network synthesis [26]. Existing works adopt different frameworks to model the translation process.

Some works utilize the encoder-decoder framework by learning abstract latent representation of the input graph through the encoder and then generating the target graph based on these hidden information through the decoder [26], [64]. For exampl, Guo et al. [26] propose a GAN-based model for graph topology transformation. The proposed GT-GAN consists of a graph translator and a conditional graph discriminator. The graph translator includes two parts: graph encoder and graph decoder. A graph convolution neural net [98] is extended to serve as the graph encoder in order to embed the input graph into node-level representations while a new graph deconvolution net is used as the decoder to generate the target graph. Zhou et al. [83] propose modeling the underlying distribution of graph structures of the input graph at different levels of granularity, and then "transferring" such hierarchical distribution from the graphs in the source domain to a unique graph in the target domain. The input graph is characterized as several coarse-grained graphs by aggregating the strongly coupled nodes with a small algebraic distance to form coarser nodes. Overall, the framework can be separated into three stages. At the first step, the coarse-grained graphs at K levels of granularity are constructed from the input graph adjacent matrix A_S. The adjacent matrix of the coarse-grained graph $A_S^{(l)} \in \mathbb{R}^{N^{(l)} \times N^{(l)}}$ at the kth layer is defined as

$$A_S^{(k)} = {P^{(k-1)}}^T \ldots {P^{(1)}}^T A_S P^{(1)} \ldots P^{(k-1)}, \tag{21}$$

where $P^{(k)} \in \mathbb{R}^{N^{(l)} \times N^{(l)}}$ is a coarse-grained operator for the kth level and $N^{(l)}$ refers to the number of nodes of the coarse-grained graph at level l. In the next stage, each coarse-grained graph at each level k will be reconstructed back into a fine graph adjacent matrix $A_T^{(k)} \in \mathbb{R}^{N^{(l)} \times N^{(l)}}$ as

$$A_T^{(k)} = {R^{(1)}}^T \ldots {R^{(k-1)}}^T A_S^{(k)} R^{(k-1)} \ldots R^{(1)}, \tag{22}$$

where $R^{(k)} \in \mathbb{R}^{N^{(l)} \times N^{(l)}}$ is the reconstruction operator for the kth level. Thus all the reconstructed fine graphs at each layer are in the same scale. Finally, these graphs are aggregated into a unique one by a linear function to get the final adjacent matrix as follows: $A_T = \sum_{k=1}^{K} w^k A_T^{(k)} + b^k$, where $w^k \in \mathbb{R}$ and $b^k \in \mathbb{R}$ are weights and bias.

3.1.2 Node-Edge Co-Transformation

Overall Problem Formulation. The problem of node-edge co-transformation (NECT) is generating the node and edge attributes of the target graph conditioning on those of the input graph. It requires that both the nodes and edges can vary during the transformation process between the source graph and the generated target graph as follows: $\mathcal{T} : G_S(\mathcal{V}_S, \mathcal{E}_S, F_S, E_S) \rightarrow G_T(\mathcal{V}_T, \mathcal{E}_T, F_T, E_T)$. In terms of the techniques on how the input graph is assimilated to generate the target graph, there are two categories: one is embedding-based and the other is editing-based.

Embedding-Based NECT. The embedding-based NECT normally encodes the source graph into latent representations containing higher-level rich information of the input graph by an encoder, which is then decoded into the target graph by a decoder, as shown in Fig. 4a [25], [84], [85], [86],

2. Purely node and edge attribute transformation have been handled in node classification or link prediction task by typical GNNs [74], [97], thus are not the focus of our survey.

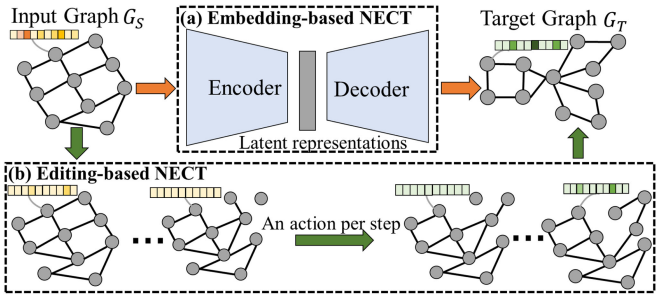


Fig. 4. Embedding-based NECT versus Editing-based NECT.

[89]. These methods are usually based on conditional VAEs [99] and conditional GANs [100].

Kaluze et al. [84] propose exploring the latent spaces of directed acyclic graphs (DAGs) and develops a neural network-based DAG-to-DAG translation model, where both the domain and the range of the target function are DAG spaces. The encoder M_{encode} is borrowed from the deep-gated DAG recursive neural network (DG-DAGRNN) [101], which generalizes stacked RNNs on sequences to DAG structures. Each layer of the DG-DAGRNN consists of gated recurrent units (GRUs), which are repeated for each node v_i. The encoder outputs an embedding $h = M_{\text{encode}}(G_S)$, which serves as the input of the DAG decoder. The decoder follows the local-based node-sequential generation style as described in Section 2.1.1. Specifically, first, the number of nodes N of the target graph is predicted by an MLP with the input of h. Also, the hidden state of the target graph is initialized with h. Then at each step, a node v_i as well as its corresponding edge set $\{e_{i,j}\}_{j<i}$ are generated based on the hidden state at each step until an end node is added to the graph or the number of nodes exceeds a predefined threshold. Following this, a general graph-to-graph model [25] is proposed by first formalizing the graph into a DAG without loss of information and utilize recurrent based model to translate this DAG. They embeds the topology of the input graph into the node representations by exerting a topology constraint, which results in a topology-flow encoder. Their decoder follows the same node sequential-based generation as proposed by You et al. [19]. There are also some embedding-based graph translation methods that represent the graph as a set of graph motifs, which are usually targeted for the task of molecule optimization [85], [86].

Editing-Based NECT. Different from the encoder-decoder framework, Editing-based NECT directly modifies the input graph iteratively to generate the target graphs [87], [88], [102], as shown in Fig. 4b. There are two ways to realize the process of editing the source graph. One is utilizing an RL agent to sequentially modify the source graph based on a formulated Markov decision process [87], [88] as described in Appendix. The modification at each step will be selected from the defined action set, including "add node," "add edge," "remove bonds" et al. The other is to update nodes and edges from the source graph synchronously in a one-shot manner through the MPNN using several iterations [102].

You et al. [87] propose the graph convolutional policy network (GCPN), a general graph convolutional network based model for goal-directed graph generation through reinforcement learning. The model is trained to optimize the domain-specific property of the source molecule through policy gradient, and acts in an environment that incorporates domain-specific rules. They define a distinct, fixed-dimension and homogeneous action space amenable to reinforcement learning, where an action is analogous to link prediction. Specifically, they first define a set of scaffold sub-graphs $\{C_1, \ldots, C_s\}$ based on the source graph. This set acts as a sub-graph vocabulary that contains the sub-graphs to be added into the target graph during graph generation. Given the modified graph G_t at step t, they define the corresponding extended graph as $G_t \cup C_i$. Based on this definition, an action can either correspond to connecting a new sub-graph C_i to a node in G_t or connecting existing nodes within graph G_t.

Guo et al. [102] propose another way which edits the source graph iteratively, through the generation process extended from *MPNN-based adjacency-based one-shot method* in Section 2.2.1 and Fig. 4c, which conducts the generation on both the node and edge attributes. The transformation process is modeled by several stages and each stage generates an immediate graph. Specifically, at each stage t, there are two paths, namely node translation and edge translation paths. In node translation path, an MLP-based *influence-function* is used for calculating the influence $I_i^{(t)}$ on each node v_i from its neighboring nodes, and another MLP-based *updating-function* is used for updating the node attribute as $F_i^{(t)}$ with the input of influence $I_i^{(t)}$. The edge translation path is constructed in the same way as the node translation path, where each edge is generated by the influence from its adjacent edges. In addition, to capture and maintain the consistent between nodes and edges in the generated graph, a spectral-based regularization is enforced into the final optimization objective.

3.1.3 Comparison of Different Sub-Categories

In this sub-section, we compare the two categories of methods in dealing with the node-edge-co-transformation (NECT). Since the comparison between different generating techniques is provided in Section 2, here we focus on the discussion regarding the relationship between the input and target graphs from three aspects: (1) *Patterns captured from input graphs*: embedding-based NECT can capture the influences from the global patterns (e.g., density or molecule energy) of the input graphs onto the target graph with a graph-level latent representation. While the editing-based NECT has the advantage in modeling the influences from the local patterns (e.g., "hub" node or ring structure) of the input graphs onto the target graphs; (2) *Interpretability*: editing-based NECT provides a more interpretable way by explicitly showing the transformation in a step-by-step fashion from the input to target graphs, which is more suitable to applications which rely on high-level confidence; While embedding-based NECT roughly connect the input and target graphs with a latent embedding which can not be semantically explained. (3) *Application scenarios*: embedding-based NECT is capable of modeling the transformation with major and sophisticated changes from the input to target graphs, while editing-based NECT is more suitable to deal with the transformation with only small change, considering the efficiency.

3.2 Conditioning on Sequence

The problem of deep graph generation conditioning on a sequence can be formalized as the deep sequence-to-graph transformation problem. It aims to generate the target graph G_T conditioning on an input sequence X. The deep sequence-to-graph problem is usually observed in domains such as NLP [90], [91] and time series mining [92], [93].

The existing methods handle semantic parsing task [90], [91] by transforming a sequence-to-graph problem into a sequence-to-sequence problem and utilizing the classical RNN-based encoder-decoder model to learn this mapping. Chen et al. [90], [91] propose a neural semantic parsing approach named *Sequence-to-Action*, which models semantic parsing as an end-to-end semantic graph generation process. Given a sentence $X = \{x_1, \ldots, x_m\}$, the *Sequence-to-Action* model generates a sequence of actions $Y = \{y_1, .., y_m\}$ for constructing the correct semantic graph. A semantic graph consists of nodes (including variables, entities, types) and edges (semantic relations), with some universal operations (e.g., argmax, argmin, count, sum, and not). To generate a semantic graph, they define six types of actions: *Add Variable Node*, *Add Entity Node*, *Add Type Node*, *Add Edge*, *Operation Function* and *Argument Action*. In this way, the generated parse tree is represented as a sequence, and the sequence-to-graph problem is transformed into a sequence-to-sequence problem. Then the attention-based sequence-to-sequence RNN model [103] with an encoder and decoder is utilized, where the encoder converts the input sequence X to a sequence of context sensitive vectors $\{b_1, \ldots, b_m\}$ using a bidirectional RNN and a classical attention-based decoder generates action sequence Y. Other methods handle the problem of Time Series Conditioned Graph Generation [92], [93]: given an input multivariate time series, the aim is to infer a target relation graph to model the underlying interrelationship between the time series and each node. Yang et al. [92] explore GANs in the conditional setting and propose the novel model of time series conditioned graph generation-generative adversarial networks (TSGG-GAN) for time series conditioned graph generation. Specifically, the generator in a TSGG-GAN adopts a variant of recurrent neural network called simple recurrent units (SRU) [104] to extract essential information from the time series, and uses an MLP to generate the directed weighted graph.

3.3 Conditioning on Semantic Context

The problem of deep graph generation conditioning on semantic context aims to generate the target graph G_T conditioning on an input semantic context, which can be usually represented as additional meta-features. The semantic context can refer to the category, label, modality or any additional information that can be intuitively represented as a vector C. The main issue is deciding where to concatenate or embed the condition representation into the generation process. As a summary, the conditioning information can be added in terms of one or multiple of the following modules: (1) the node state initialization module, (2) the message passing process for MPNN-based decoding, and (3) the conditional distribution parameterization for sequential generating.

Yang et al. [94] propose a novel unified model of graph variational generative adversarial nets, where the conditioning semantic context is inputted into the node state initialization module. Specifically, in the generation process, they first model the embedding Z_i of each node with separate latent distributions. Then, a conditional graph VAE (CGVAE) can be directly constructed by concatenating the condition vector C to each node latent representation Z_i to get the updated node latent representation $\hat{Z}_i$. Thus, the distribution of the individual edge $A_{i,j}$ is assumed as a Bernoulli distribution, which is parameterized by the value $\hat{A}_{i,j}$ and is calculated as $\hat{A}_{i,j} = \text{Sigmoid}(f(\hat{Z}_i)^T f(\hat{Z}_j))$, where $f(\cdot)$ is constructed by a few fully connected layers. Li et al. [45] propose a conditional deep graph generative model that adds the semantic context information into the initialized latent representations Z_i at the beginning of the decoding process.

Li et al. [96] add the context information C into the message passing module in its MPNN-based decoding process. Specifically, they parameterize the decoding process as a Markov process and generate the graph by iteratively refining and updating from the initialized graph. At each step t, an action is conducted based on the current node hidden states $H^t = \{h_1^t, \ldots, h_N^t\}$. To calculate $h_i^t \in \mathbb{R}^L$ (L denotes the length of the representation) for node v_i in the intermediate graph G_t after each updating of the graph, they utilize message passing network with node message propagation. Thus the context information $C \in \mathbb{R}^K$ is added to the operation of the MPNN layer as follows:

$$h_i^t = Wh_i^{t-1} + \Phi \sum\nolimits_{v_j \in N(v_j)} h_j^{t-1} + \Theta C, \tag{23}$$

where $W \in R^{L \times L}$, $\Theta \in R^{L \times L}$ and $\Phi \in R^{K \times L}$ are all learnable weights vectors and K denotes the length of the semantic context vector.

4 Evaluation Metrics for Deep Graph Generation

Evaluating the generated graphs as well as the learnt distribution of graphs are challenging and critical tasks for deep generative models in graph generation problem due to two major reasons: 1) Different from conventional prediction problems where merely deterministic predictions need to be evaluated, deep graph generation requires the evaluation of the learnt distributions. 2) Graph structured data is much more difficult to evaluate than simple data with matrix/vector structures or semantic data such as images and texts. Thus, we summarize the typical evaluation metrics in evaluating deep generative models for graph generation as shown in Table 3. We first provide the metrics that can be used both for unconditional and conditional deep graph generation, and then introduce the metrics that are specially designed for conditional deep graph generation.

4.1 General Evaluation for Deep Graph Generation

To evaluate the quality of the generated graphs, existing literature covers three categories of evaluation metrics, namely statistics-based, classifier-based, and intrinsic-quality-based evaluations. The first two evaluation categories require comparison between the generated graph set and

TABLE 3
Evaluation Metrics for Deep Generative-Based Methods

Type		*Evaluation feature*
General	Statistics-based	Average KLD MMD
	Classifier-based	Accuracy-based FID-based
	Intrinsic-quality-based	Validity Uniqueness Novelty
Condition-specialized	Graph property-based Mapping-relationship-based	

real graph set, while the intrinsic-quality evaluation directly measure the properties of the generated graph.

4.1.1 Statistics-Based

In statistics-based evaluation, the quality of the generated graphs is accessed by computing the distance between the graph statistic distribution of real graphs and generated graphs. We first introduce seven typical graph statistics that measure different properties of graphs and, thereafter introduce the metrics that measure the distance between two distributions regarding different graph statistics.

There are seven typical graph statistics that are used in existing literature, which are summarized as follows: (1) *Node degree distribution:* the empirical node degree distribution of a graph, which could encode its local connectivity patterns. (2) *Clustering coefficient distribution:* the empirical clustering coefficient distribution of a graph. Intuitively, the clustering coefficient of a node is calculated as the ratio of the potential number of triangles the node could be part of to the actual number of triangles the node is part of. (3) *Orbit count distribution*; the distribution of the counts of node 4-orbits of a graph. Intuitively, an orbit count specifies how many of these 4-orbits substructures the node is part of. This measure is useful in understanding if the model is capable of matching higher-order graph statistics, as opposed to node degree and clustering coefficient, which represent measures of local (or close to local) proximity. (4) *Largest connected component:* the size of the largest connected component of the graphs. (5) *Triangle count:* the number of triangles counted in the graph. (6) *Characteristic path length:* the average number of steps along the shortest paths for all node pairs in the graph. (7) *Assortativity:* the Pearson correlation of degrees of connected nodes in the graph.

The first three graph statistics are about distributions of each graph and are always represented as a vector, while the last four graph statistics are represented as scalar values of each graph. Therefore, to evaluate the distance between two sets of graphs in terms of the above distribution statistics, two major metrics are usually utilized in existing literature, which are introduced as follows.

Average Kullback-Leibler Divergence. Considering that each graph set has a set of distributions in terms of a graph property x, we first calculate the average distribution of the whole set. To get the average distribution of a graph set, the vectors of counts of the property x of all the graphs are first concatenated. Then the probability densities of the graph property x is calculated based on this concatenated vector as the average node degree distribution. Fianlly, the Kullback-Leibler divergence (KL-D [105]) between the average node degree distribution of the generated graph set $P_{\text{ave}}(x)$ and that of the real graph set $Q_{\text{ave}}(x)$ is calculated as

$$KL - D(P_{\text{ave}}, Q_{\text{ave}}) = -\sum\nolimits_{x \sim P_{\text{ave}}} P_{\text{ave}}(x) log(Q_{\text{ave}}(x)/P_{\text{ave}}(x)).$$

Maximum Mean Discrepancy (MMD) [106]. First, the squared MMD between the graph statistics distribution of the generated graph set P and that of the real graph set Q can be derived as

$$MMD(P,Q) = \mathbb{E}_{x,y\sim P}[k(x,y)] + \mathbb{E}_{x,y\sim Q}[k(x,y)] - 2\mathbb{E}_{x\sim P, y\sim Q}[k(x,y)], \quad (24)$$

where x, y refer to the graph statistics that are sampled from the two distributions. The kernel $k(*)$ is designed as follows:

$$k(x,y) = exp(W(x,y)/2\sigma^2), \quad (25)$$

where σ refers to the standard deviation of P or Q. Considering the sampled graph statistics are also two distributions, thus, $W(x,y)$ is defined as the Wasserstein distances (WD)

$$W(x,y) = inf\, \gamma \in \prod(x,y)\mathbb{E}_{(i,j)\sim\gamma}[\| i - j \|], \quad (26)$$

where $\prod(x,y)$ is the set of all measures whose marginals are x and y respectively.

Distance Metrics for Scalar-Valued Statistics. The calculation of distance between two sets of graphs in terms of the scalar-valued statistic is much easier than that of distribution statistics. There are two major ways: (1) calculating the difference between the averaged value of the scalar-valued statistic of the generated graph set and that of the real graph set; (2) calculating the distance between the distribution of the scalar-valued statistic of the generated graph set and that of the real graph set. Many distance metrics can be used, such as KL-D, Jensen-Shannon distances (JS), and the Hellinger distance (HD).

4.1.2 Classifier-Based

Classifier-based evaluation typically utilizes a graph classifier to evaluate whether the generated graphs follows the same distribution as the real graphs without explicitly defining the graph statistics. Typically, a classifier is trained on the set of real graphs and is tested on the set of generated graphs. It only could be utilized when multiple graph generative models are trained for generating multiple types of graphs, respectively. Here we introduce two existing classifier-based evaluations [27] that are based on graph isomorphism network (GIN) [107] as follows.

Accuracy-Based. First, a GIN is pre-trained on the training set consisting of multiple types of graphs previously used for training the generative model. Then for each type of generated graph, the classification accuracy of classifying this type of generated graphs based on the trained GIN is the final evaluation metric.

Fréchet Inception Distance (FID)-Based. FID computes the distance in the embedding space between two multivariate Gaussian distributions fitted to a generated set and a test set. A lower FID value indicates better generation quality and diversity. For each type of graph, first, the generated and real graphs in the testing set are inputted into the pre-trained GIN to get the graph embeddings. Then the means μ_{G} and covariance matrices $\sum_{\mathrm{G}}$ of the embeddings of the generated graph set, and the means μ_{R} and covariance matrices $\sum_{\mathrm{R}}$ of real graphs are estimated. Finally, the FID metric for this type of graphs is computed as follows:

$$FID = \| \mu_{\mathrm{G}} - \mu_{\mathrm{R}} \|_2^2 + \mathrm{Tr}\left(\sum\nolimits_{\mathrm{G}} + \sum\nolimits_{\mathrm{R}} - 2\left(\sum\nolimits_{\mathrm{G}} \sum\nolimits_{\mathrm{R}} \right)^{\frac{1}{2}} \right),$$

where $\mathrm{Tr}(\cdot)$ refers to the trance of a matrix.

4.1.3 Intrinsic-Quality-Based

Besides the evaluation by measuring the similarity between the real and generated graphs, there are three additional metrics that directly evaluate the quality of the generated graphs: their validity, uniqueness and novelty.

Validity. Since sometimes the generated graphs are required to preserve some properties, it is straightforward to evaluate them by judging whether they satisfy such requirements, such as the following: (1) Cycles graphs/Tree graphs: Cycles and trees are graphs that have obvious structural properties and the validity is calculated as what percentage of generated graphs are actually cycles or trees [45]. (2) Molecule graphs: Validity for molecule generation is the percentage of chemically valid molecules based on some domain specific rules [18].

Uniqueness. Ideally, high-quality generated graphs should be diverse and similar, but not identical. Thus, uniqueness is utilized to capture the diversity of generated graphs [18], [45], [48], [49], [63]. To calculate the uniqueness of a generated graph, the generated graphs that are sub-graph isomorphic to some other generated graphs are first removed. The percentage of graphs remaining after this operation is defined as Uniqueness. For example, if the model generates 100 graphs, all of which are identical, the uniqueness is $1/100 = 1\%$.

Novelty. Novelty measures the percentage of generated graphs that are not sub-graphs of the training graphs and vice versa [48], [49], [63]. Note that identical graphs are defined as graphs that are sub-graph isomorphic to each other. In other words, novelty checks if the model has learned to generalize unseen graphs.

4.2 Evaluation for Conditional Deep Graph Generation

In addition to the above general evaluation metrics for graph generation, for conditional deep generative models for graph generation, some additional evaluation metrics can be involved, including: graph-property-based and mapping-relationship-based evaluations.

4.2.1 Graph-Property-Based

Considering that each of generated graph can have its associated real graph as label in the conditional graph generation task, we can directly compare each generated graph to its label graph by measuring their similarity or distance based on some graph properties or kernels, such as the following: (1) random-walk kernel similarity by using the random-walk based graph kernel [108]; (2) combination of Hamming and Ipsen-Mikhailov distances(HIM) [109]; (3) spectral entropies of the density matrices; (4) eigenvector centrality distance [110]; (5) closeness centrality distance [111]; (6) Weisfeiler Lehman kernel similarity [112]; (7) Neighborhood Sub-graph Pairwise Distance Kernel [48] by matching pairs of sub-graphs with different radii and distances.

4.2.2 Mapping-Relationship-Based

Mapping-relationship-based evaluation measures whether the learned relationship between the conditions and the generated graphs is consistent with the true relationship between the conditions and the real graphs.

Explicit Mapping Relationship. In the situation where the true relationship between the input conditions and the generated graphs is known in advance, the evaluation can be conducted as follows: (1) When the condition is a category label, we can examine whether the generated graph falls into the conditional category by utilizing a graph classifier [22], [25]. Specifically, the real graphs are used to train a classifier and the classifier is used to classify the generated graphs. Then the accuracy is calculated as the percentage of the predicted categories that are the same as the input condition. (2) When the condition is a graph, where the task is to change some properties of the input graph, we can quantitatively compare the property scores of the generated and input graphs to see if the change indeed meets the requirement. For example, one can compute the improvement of *logP* scores of the optimized molecule in molecule optimization task [87].

Implicit Mapping Relationship. Regarding the deep graph translation problem, which is introduced in Section 3.1, sometimes, the underlying patterns of the mapping from the input graphs to the real target graphs are implicit and complex to define and measure. Thus, a classifier-based evaluation metric can be utilized [26]. By regarding the input and target graphs as two classes, it assumes that a classifier that is capable of distinguishing the generated target graphs would also succeed in distinguishing the real target graphs from the input graphs. Specifically, a graph classifier is first trained based on the input and generated target graphs. Then this trained graph classifier is tested to classify the input graph and real target graphs, and the results will be used as the evaluation metrics.

5 Applications

Deep generative models for graph generation is a very active research domain with a continuously increasing number of applications being proposed, including important topics such as molecule optimization and generation, semantic parsing in NLP, code modeling, and pseudo-industrial SAT instance generation.

5.1 Molecule Generation

Molecule generation is a challenging mathematical and computational problem in drug discovery and material

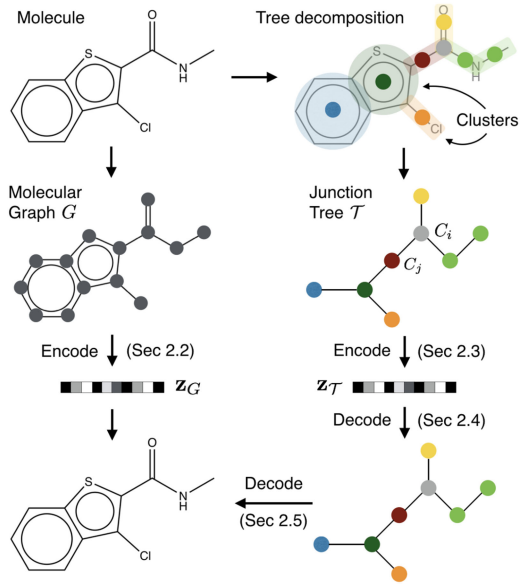


Fig. 5. Framework overview of JT-VAE [28]: A molecular graph G is first decomposed into its junction tree $\mathcal{T}_G$, where each colored node in the tree represents a substructure in the molecule. Then both the tree and graph are encoded into their latent embeddings z_T and z_G. To decode, the junction tree is first reconstructed from z_T.

science; its aim is to design novel molecules under a range of chemical properties. Any small perturbation in the chemical structure may result in a large variation in the desired molecular property. Besides, the space of valid molecules quickly becomes prohibitively huge and complex as the number of combinatorial permutations of atoms and bonds grows. Currently, most drugs are hand-crafted by human experts in chemistry and pharmacology. The recent advances of deep generative models for graph generation has opened a new research direction by treating the molecule as a graph with atoms as nodes and bonds as edges, with the potential to learn these molecular' generative representation for novel molecule generation to ensure chemical validity and efficiency [18], [28], [41], [44], [57], [113].

Representative Work. Junction Tree VAE (JT-VAE) [28] formalizes the molecular structures generation task into an unconditional graph generation problem, where each atom in a molecule is a node in the graph and the bonds between atoms are represented as edges. JT-VAE adopts a motif-sequence-based generation approach, one of a number of sequential-based generating techniques, to generates a molecular graph by sequentially expanding a generated molecule by adding a valid chemical substructure in each step. Fig. 5 shows a backbone VAE-based generative model consisting of two encoders and decoders. Here, the molecular graph G is first decomposed into its junction tree $\mathcal{T}_G$, where each colored node in the tree represents a substructure in the molecule. Then both the tree and graph are encoded into their latent embeddings z_T and z_G. To decode the molecule, first step is to reconstruct the junction tree from z_T, and then assemble nodes in the tree to return to the original molecule.

5.2 Protein Structure Modeling

Proteins are massive molecules that can be characterized as one of the multiple long chains of amino acids. Analyzing

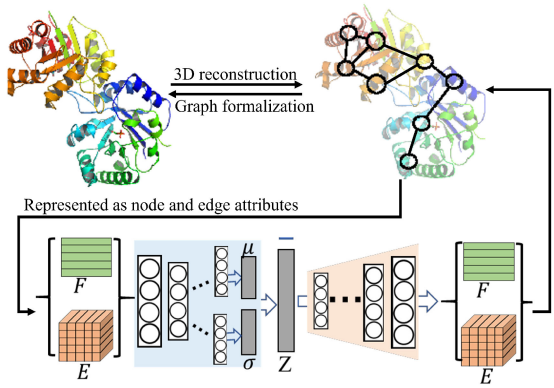


Fig. 6. CO-VAE for protein structure modeling [7]: a protein graph represents the mutual distance between each pair of amino acids. Node and edge attributes are input into the encoder to learn the distribution of the latent embedding.

the structure and function of proteins is a key part of understanding biological properties at the molecular and cellular level. Current computational modeling methods for protein design are slow and often require human oversight and intervention, which are often biased and incomplete. Inspired by recent momentum in deep graph generative models, some works [7], [8], [21], [114], [115] demonstrate the potential of deep graph generative modeling for fast generation of new, viable protein structures.

Representative Work. Guo et al. [7] proposed a contact VAE (CO-VAE) to generate functionally relevant three-dimensional protein structures. Here, the protein structure is formalized as a graph where each amino acid is a node and the physical distance between two amino acids determines the existence of an edge based on a pre-defined threshold. A graph generative model VAE is utilized to model and generate the graph by following the adjacent-matrix-based one-shot generating technique, where the node attributes and adjacent matrix of graph are generated in a single shot. As shown in Fig. 6, a protein structure is first represented by a graph that consists of a node attribute matrix and edge attribute tensor. These two components are then input into the encoder of VAE to learn the distribution of the latent embedding of the graph. In the decoder, the node and edge attributes are generated based on the sampled latent embedding and can then be recovered to yield the protein based on a 3D reconstruction technique.

5.3 Semantic Parsing

Semantic parsing problem is about mapping the natural language information to its logical forms, namely abstract meaning representation (AMR). Traditional semantic parsers are usually based on compositionally and manually designed grammar to create the structure of AMR, and used lexicons for semantic grounding, which is time-consuming and heuristic. Recent works develops neural semantic parser with sequence-to-sequence models [116], [117], which, however, only consider the word sequence information and ignore other rich syntactic information. Because AMR are naturally structured objects (e.g., tree structures), semantic AMR parsing methods based on deep graph generative models are deemed as promising [11], [29], [90], [91]. These methods represent the semantics of a sentence as a

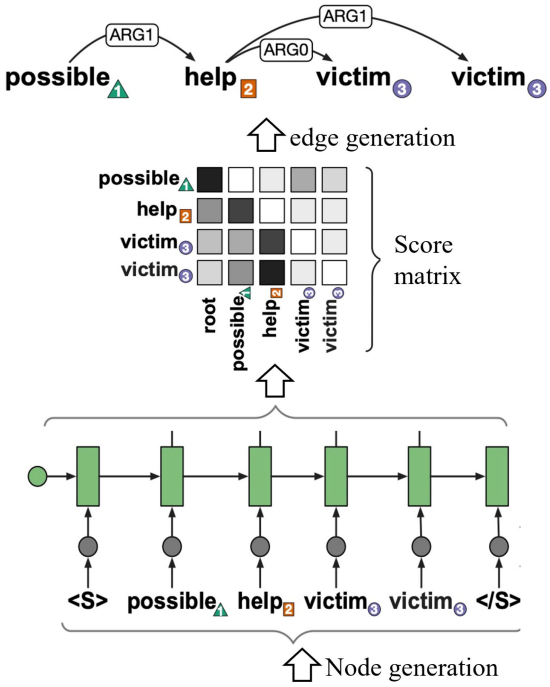


Fig. 7. A two-stage AMR parsing process for a sequence-to-graph problem [11]: node prediction is to generate nodes based on the input sequence of tokens and edge generation generates the edges by sampling from the score matrix calculated based on the node representations.

semantic graph (i.e., a sub-graph of a knowledge base) and treat semantic parsing as a semantic graph matching/generation process.

Representative Work. Zhang et al. [11] formalized the AMR parsing as a graph generation problem conditioned on sequence, where the input is the sequence of tokens from a target sentence and the output its AMR graph. In the AMR graph, a node denotes to a word in the sentence and a predicted edge represents the semantic relationship between two words. This work is an edge-list-based one shot generation method, where the edges are generated based on pairs of node representations. As shown in Fig. 7, the whole process consists of two stages: node and edge prediction. The node prediction utilizes an RNN-based generative models to generate the nodes selected from the tokens in the sentence. For the second stage (i.e., the edge prediction), a score matrix that measures the probability of the edge existence is learnt based on the representation vectors of each pair of nodes, after which the edge is generated by sampling from the score matrix. This end-to-end deep graph generation techniques for semantic parsing has demonstrated a powerful ability for automatically capturing semantic information.

5.4 Code Modeling

Code modelling considers both hard syntactic and semantic constraints in generating natural programming code, which can make the development of source code easier, faster, and less error-prone. Early works in this area have shown that approaches from natural language processing can be applied successfully to the source code. However, though

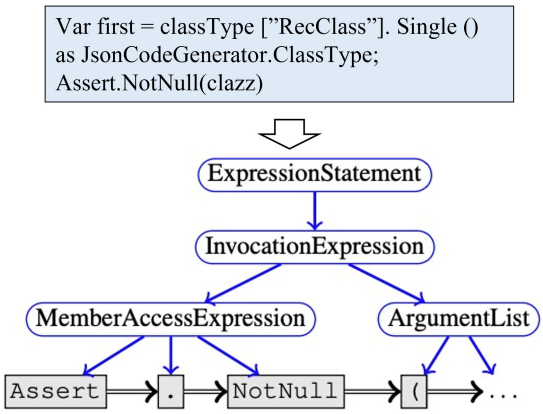


Fig. 8. Representing a program as an abstract syntax tree [9], [120]: each node refers to a construct occurring in the codes and the edges denote to the semantic relationships.

these methods can generate programs that satisfy some formal specifications, they cannot generate realistic-looking and valid programs. Since *program graphs* have been shown to have the ability to encode semantically meaningful representations of programs, deep graph generative models have shown promising capability in modeling small but semantic programs generation [9], [10], [118], [119].

Representative Work. Brockschmidt et al. [9] formalized the code modeling as a graph structure generation problem, where the source code is represented by an abstract syntax tree, as shown in Fig. 8. In this abstract syntax tree, each node refers to a construct occurring in the codes and the edges denote to the semantic relationships. The generation process is a rule-based sequentially generating techniques, where the code is represented as an abstract syntax tree (AST), which incorporates rich structural information. An AST is then generated by expanding one node at a time using production rules from the underlying programming language grammar. This simplifies the code generation task to a sequence of sampling problems, in which an appropriate production rule must be sampled based on the partial AST generated so far.

5.5 Pseudo-Industrial SAT Instance Generation

The problem of pseudo-industrial Boolean Satisfiability (SAT) instance generation is about generating artificial SAT problems that display the same characteristics as their real-world counterparts. Generating large amounts of SAT instances is important in developing and evaluating practical SAT solvers, which historically relies on extensive empirical testing on a large amount of SAT instances. Prior works addressing this problem relied on hand-crafted algorithms, but have difficult in simultaneously capturing a wide range of characteristics exhibited by real-world SAT instances [121], [122]. Thus, it is promising to represent SAT formulas as graphs, thus recasting the original problem as a deep graph generation task [123], [124].

Representative Work. G2SAT [123] formalizes the SAT generation task as a graph generation problem by representing the SAT as a bipartite graph, where each node represents either a literal or a clause, with an edge denoting the occurrence of a literal in a clause representing a disjunction operation. In general, the generation process is a motif-

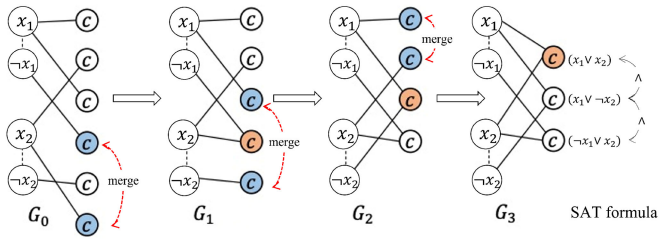


Fig. 9. An overview of the G2SAT model [123]: In each step, two clause nodes are merged into a single clause node. A GCN-based classifier that captures the bipartite graph structure is used to sequentially decide which nodes to merge.

sequence-based generating style where a new motif is added to the partially generated graph in each step. The motifs refer to the trees that are split from the existing training bipartite graphs. As shown in Fig. 9, while generating, G2SAT generates a bipartite graph by starting with a set of motifs. In each step, a new motif is added by merging one of its clause nodes with an existing node. At last, all the conjunction clauses are combined with conjunction operations to recover the SAT formula.

6 Future Opportunities

As a fast-developing, promising domain, there are still many open challenges in the domain of deep generative models for graph generation. In this section, we highlight a number of open challenges for further research.

Scalability. Existing deep generative models typically have super-linear time complexity to the number of nodes and cannot scale well to large networks. Only few methods have linear time complexity of $O(N)$ [19], [20], [27], [48], [70] and $O(M)$ [113], where N is the number of nodes and M is the number of edges. Consequentially, most existing works merely focus on small graphs, typically with dozens to thousands of nodes [2], [22], [45], [54], [57], [73], [87]. However, many real-world networks are large, with millions to billions of nodes [48], such as the Internet, biological neuronal networks, and social networks. It is important for any generative model to scale to large graph.

Validity Constraint. Many real-world networks are constrained by specific validity requirements [55]. For example, in molecular graphs, the number of bonding-electron pairs cannot exceed the valency of an atom. In protein interaction networks, two proteins may be connected only when they belong to the same or correlated gene ontology terms. Graph-topological constraints are challenging to enforce during the model training process. Intuitive ways include designing heuristic and customized algorithms to ensure the validity of generated graphs. For example, Dai et al. [10] further apply attribute grammar as a constraint in the parse-tree generation, a step toward semantic validity. Some recent works started to construct a more generic framework under constrained optimization scenario, which minimizes training loss under graph validity constraints [55]. However, as such constraints are typically discrete and non-differentiable, they need to be approximated with a smooth relaxation which introduces errors and cannot preclude all the invalid topologies.

Interpretability. When we learn the underlying distribution of complex structured data, i.e., graphs, learning interpretable representations of data that expose semantic meaning is very important [125]. For example, it is highly beneficial to identify which latent variable(s) control(s) which specific properties (e.g., molecule mass) of the generated graphs (e.g., molecules). It is also useful to disentangle local generative dependencies among different sub-graphs. However, existing works on this topic only focus on graph embedding but not generation [126], [127]. For example, Stoehr et al. [128] demonstrates the potential of latent variable disentanglement in graph deep learning for unsupervised discovery of generative parameters of random and real-world graphs. Investigations on graph generation are still open problems without existing works except very recently published ones [129], [130], [131].

Beyond Training Data. Deep generative models are data-driven models based on training data. The novelty of the generated graphs are highly desired yet usually restricted by training data and model properties (e.g., mode collapse of generative adversarial nets). To address such issues, attempts in the domain of images modified the attribute of a generated image by adding a learned vector on its latent code [132] or by combining the latent code of two images [133]. Additional works have been developed for inserting extra control in the image generation [132] with additional labels corresponding to key factors such as object size and facial expression. However, works on graph generation that could require very different technique sets than image generation are lacking.

Dynamic Graphs. Existing deep graph generative models typically focus on static graphs but many graphs in the real-world are dynamic, and their node attributes and topology can evolve over time, such as social network, mobility network, and protein folding. Representation learning for dynamic graphs is a hot domain, but it only focuses on graph embedding instead of generation. Modeling and understanding the generation of dynamic graphs have not been explored. Therefore, additional problems such as jointly modeling temporal and graph patterns and temporal validity constraints need to be addressed.

7 Conclusion

In this survey paper, we provides a systematic review of deep generative models for graph generation. We present a taxonomy of deep graph generative models based on problem settings and techniques details, followed by a detailed introduction, comparison, and discussion about them. We also conduct a systematic review of the evaluation measures of deep graph generative models, including the general evaluation metrics for both unconditional and conditional graph generation. After that, we summarized popular applications in this domain.

References

[1] A.-L. Barabási et al., *Network Science*. Cambridge, U.K.: Cambridge Univ. Press, 2016.
[2] A. Grover, A. Zweig, and S. Ermon, "Graphite: Iterative generative modeling of graphs," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, pp. 2434–2444.

[3] H. Wang et al., "GraphGAN: Graph representation learning with generative adversarial nets," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 2508–2515.

[4] C. Tran et al., "DeepNC: Deep generative network completion," 2019, *arXiv:1907.07381*.

[5] S. Wang, X. Guo, and L. Zhao, "Deep generative model for periodic graphs," 2022, *arXiv:2201.11932*.

[6] S. Ranu and A. K. Singh, "GraphSig: A scalable approach to mining significant subgraphs in large graph databases," in *Proc. IEEE 25th Int. Conf. Data Eng.*, 2009, pp. 844–855.

[7] X. Guo et al., "Generating tertiary protein structures via an interpretative variational autoencoder," 2020, *arXiv:2004.07119*.

[8] Y. Du, X. Guo, A. Shehu, and L. Zhao, "Interpretable molecular graph generation via monotonic constraints," in *Proc. SIAM Int. Conf. Data Mining*, 2022, pp. 73–81.

[9] M. Brockschmidt et al., "Generative code modeling with graphs," in *Proc. Int. Conf. Learn. Representations*, 2018.

[10] H. Dai et al., "Syntax-directed variational autoencoder for structured data," in *Proc. Int. Conf. Learn. Representations*, 2018.

[11] S. Zhang et al., "AMR parsing as sequence-to-graph transduction," in *Proc. 57th Annu. Meeting Assoc. Comput. Linguistics*, 2019, pp. 80–94.

[12] M. Newman, *Networks*. Oxford, U.K.: Oxford Univ. Press, 2018.

[13] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Rev. Modern Phys.*, vol. 74, no. 1, 2002, Art. no. 47.

[14] J. Leskovec et al., "Kronecker graphs: An approach to modeling networks," *J. Mach. Learn. Res.*, vol. 11, no. Feb, pp. 985–1042, 2010.

[15] G. Robins et al., "Recent developments in exponential random graph (p*) models for social networks," *Social Netw.*, vol. 29, no. 2, pp. 192–215, 2007.

[16] P. Erdös et al., "On random graphs," *Publicationes Mathematicae*, vol. 6, no. 26, pp. 290–297, 1959.

[17] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world'networks," *Nature*, vol. 393, no. 6684, 1998, Art. no. 440.

[18] M. Popova et al., "MolecularRNN: Generating realistic molecular graphs with optimized properties," 2019, *arXiv:1905.13372*.

[19] J. You et al., "GraphRNN: Generating realistic graphs with deep auto-regressive models," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 5708–5717.

[20] D. Bacciu, A. Micheli, and M. Podda, "Graph generation by sequential edge prediction," in *Proc. Eur. Symp. Artif. Neural Netw.*, 2019, pp. 95–100.

[21] N. Anand and P. Huang, "Generative modeling for protein structures," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 7494–7505.

[22] S. Fan and B. Huang, "Labeled graph generative adversarial networks," 2019, *arXiv:1906.03220*.

[23] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2013, *arXiv:1312.6114*.

[24] I. Goodfellow et al., "Generative adversarial nets," in *Proc. 27th Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 2672–2680.

[25] M. Sun and P. Li, "Graph to graph: A topology aware approach for graph structures learning and generation," in *Proc. 22nd Int. Conf. Artif. Intell. Statist.*, 2019, pp. 2946–2955.

[26] X. Guo, L. Wu, and L. Zhao, "Deep graph translation," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Mar. 17, 2022, doi: 10.1109/TNNLS.2022.3144670.

[27] C.-C. Liu et al., "Auto-regressive graph generation modeling with improved evaluation methods," in *Proc. 33rd Int. Conf. Neural Inf. Process. Syst. Workshop*, 2019.

[28] W. Jin, R. Barzilay, and T. Jaakkola, "Junction tree variational autoencoder for molecular graph generation," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 2323–2332.

[29] C. Lyu and I. Titov, "AMR parsing as graph prediction with latent alignment," in *Proc. 56th Annu. Meeting Assoc. Comput. Linguistics*, 2018, pp. 397–407.

[30] A. Oussidi and A. Elhassouny, "Deep generative models: Survey," in *Proc. IEEE Int. Conf. Intell. Syst. Comput. Vis.*, 2018, pp. 1–8.

[31] I. Sutskever, J. Martens, and G. E. Hinton, "Generating text with recurrent neural networks," in *Proc. 28th Int. Conf. Mach. Learn.*, 2011, pp. 1017–1024.

[32] G. Papamakarios, T. Pavlakou, and I. Murray, "Masked autoregressive flow for density estimation," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 2338–2347.

[33] A. Bonifati, I. Holubová, A. Prat-Pérez, and S. Sakr, "Graph generators: State of the art and open challenges," *ACM Comput. Surv.*, vol. 53, no. 2, pp. 1–30, 2020.

[34] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowl.-Based Syst.*, vol. 151, pp. 78–94, 2018.

[35] Z. Wu et al., "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 4–24, Jan. 2021.

[36] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 1, pp. 249–270, Jan. 2022.

[37] F. Faez, Y. Ommi, M. S. Baghshah, and H. R. Rabiee, "Deep graph generators: A survey," *IEEE Access*, vol. 9, pp. 106675–106702, 2021.

[38] Y. Zhu et al., "A survey on deep graph generation: Methods and applications," 2022, *arXiv:2203.06714*.

[39] M. Khodayar, Y. Zhang, and J. Wang, "Deep generative graph distribution learning for synthetic power grids," 2019, *arXiv:1901.09674*.

[40] L. D'Arcy, P. Corcoran, and A. Preece, "Deep q-learning for directed acyclic graph generation," 2019, *arXiv:1906.02280*.

[41] M. Zhang et al., "D-VAE: A variational autoencoder for directed acyclic graphs," in *Proc. 33rd Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 1586–1598.

[42] S.-Y. Su, H. Hajimirsadeghi, and G. Mori, "Graph generation with variational recurrent neural network," 2019, *arXiv:1910.01743*.

[43] R. Assouel et al., "DEFactor: Differentiable edge factorization-based probabilistic graph generation," 2018, *arXiv:1811.09766*.

[44] J. Lim et al., "Scaffold-based molecular design using graph generative model," *Chem. Sci.*, vol. 11, no. 4, pp. 1153–1164, 2020.

[45] Y. Li et al., "Learning deep generative models of graphs," 2018, *arXiv:1803.03324*.

[46] Q. Liu et al., "Constrained graph variational autoencoders for molecule design," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 7795–7804.

[47] S. Kearnes, L. Li, and P. Riley, "Decoding molecular graph embeddings with reinforcement learning," in *Proc. Int. Conf. Workshop. Mach. Learn.*, 2019.

[48] N. Goyal, H. V. Jain, and S. Ranu, "GraphGen: A scalable approach to domain-agnostic labeled graph generation," in *Proc. Web Conf.*, 2020, pp. 1253–1263.

[49] D. Bacciu, A. Micheli, and M. Podda, "Edge-based sequential graph generation with recurrent neural networks," *Neurocomputing*, vol. 416, pp. 177–189, 2020.

[50] R. Liao et al., "Efficient graph generation with graph recurrent attention networks," in *Proc. 33rd Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 4257–4267.

[51] M. Podda, D. Bacciu, and A. Micheli, "A deep generative model for fragment-based molecule generation," in *Proc. Int. Conf. Artif. Intel. Stat.*, 2020, pp. 2240–2250.

[52] X. Gu, "Explore deep graph generation," 2019. [Online]. Available: https://laoreja.github.io/projects/cs224w_report.pdf

[53] M. J. Kusner, B. Paige, and J. M. Hernández-Lobato, "Grammar variational autoencoder," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 1945–1954.

[54] M. Simonovsky and N. Komodakis, "GraphVAE: Towards generation of small graphs using variational autoencoders," in *Proc. Int. Conf. Artif. Neural Netw.*, 2018, pp. 412–422.

[55] T. Ma, J. Chen, and C. Xiao, "Constrained generation of semantically valid graphs via regularizing variational autoencoders," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 7113–7124.

[56] S. Pölsterl and C. Wachinger, "Likelihood-free inference and generation of molecular graphs," 2019, *arXiv:1905.10310*.

[57] N. De Cao and T. Kipf, "MolGAN: An implicit generative model for small molecular graphs," in *Proc. Int. Conf. Mach. Learn. Workshop*, 2018.

[58] X. Bresson and T. Laurent, "A two-step graph convolutional decoder for molecule generation," 2019, *arXiv:1906.03412*.

[59] M. Guarino, A. Shah, and P. Rivas, "DiPol-GAN: Generating molecular graphs adversarially with relational differentiable pooling," 2017. [Online]. Available: https://www.reev.us/pdfs/guarino2019dipol.pdf

[60] D. Flam-Shepherd, T. Wu, and A. Aspuru-Guzik, "Graph deconvolutional generation," 2020, *arXiv:2002.07087*.

[61] C. Niu et al., "Permutation invariant graph generation via score-based generative modeling," in *Proc. Int. Conf. Artif. Intell. Stat.*, 2020, pp. 4474–4484.

[62] S. Honda et al., "Graph residual flow for molecular graph generation," 2019, *arXiv:1909.13521*.

[63] K. Madhawa et al., "GraphNVP: An invertible flow model for generating molecular graphs," 2019, *arXiv:1905.11600*.

[64] Y. Gao, X. Guo, and L. Zhao, "Local event forecasting and synthesis using unpaired deep graph translations," in *Proc. 2nd ACM SIGSPATIAL Workshop Analytics Local Events News*, 2018, pp. 1–8.

[65] A. Bojchevski et al., "NetGAN: Generating graphs via random walks," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 609–618.

[66] A. Gamage et al., "Multi-motifGAN (MMGAN): Motif-targeted graph generation and prediction," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2020, pp. 4182–4186.

[67] L. Zhang, "STGGAN: Spatial-temporal graph generation," in *Proc. 27th ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst.*, 2019, pp. 608–609.

[68] V. Caridá et al., "Can NetGAN be improved on short random walks?," in *Proc. IEEE 8th Braz. Conf. Intell. Syst.*, 2019, pp. 663–668.

[69] T. N. Kipf and M. Welling, "Variational graph auto-encoders," 2016, *arXiv:1611.07308*.

[70] C. Shi et al., "GraphAF: A flow-based autoregressive model for molecular graph generation," in *Proc. Int. Conf. Learn. Representations*, 2019.

[71] D. Zou and G. Lerman, "Encoding robust representation for graph generation," in *Proc. Int. Joint Conf. Neural Netw.*, 2019, pp. 1–9.

[72] J. Liu et al., "Graph normalizing flows," in *Proc. 33rd Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 13556–13566.

[73] G. Salha et al., "Gravity-inspired graph autoencoders for directed link prediction," in *Proc. 28th ACM Int. Conf. Inf. Knowl. Manage.*, 2019, pp. 589–598.

[74] J. Gilmer et al., "Neural message passing for quantum chemistry," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 1263–1272.

[75] C. M. Bishop, *Pattern Recognition and Machine Learning*, Berlin, Germany: Springer, 2006.

[76] X. Yan and J. Han, "gSpan: Graph-based substructure pattern mining," in *Proc. IEEE Int. Conf. Data Mining*, 2002, pp. 721–724.

[77] F. Scarselli, M. Gori, and A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 61–80, Jan. 2009.

[78] M. Rarey and J. S. Dixon, "Feature trees: A new molecular similarity measure based on tree matching," *J. Comput.-Aided Mol. Des.*, vol. 12, no. 5, pp. 471–490, 1998.

[79] J. Degen et al., "On the art of compiling and using'drug-like-'chemical fragment spaces," *ChemMedChem: Chem. Enabling Drug Discov.*, vol. 3, no. 10, pp. 1503–1507, 2008.

[80] E. Jang, S. Gu, and B. Poole, "Categorical reparametrization with gumble-softmax," in *Proc. Int. Conf. Learn. Representations*, 2017.

[81] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," in *Proc. Int. Conf. Learn. Representations*, 2017.

[82] K. Do, T. Tran, and S. Venkatesh, "Graph transformation policy network for chemical reaction prediction," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2019, pp. 750–760.

[83] D. Zhou et al., "Misc-GAN: A multi-scale generative model for graphs," *Front. Big Data*, vol. 2, 2019, Art. no. 3.

[84] M. C. D. P. Kaluza, S. Amizadeh, and R. Yu, "A neural framework for learning DAG to DAG translation," in *Proc. Int. Conf. Neural Inf. Process. Syst. Workshop*, 2018.

[85] W. Jin et al., "Learning multimodal graph-to-graph translation for molecular optimization," in *Proc. Int. Conf. Learn. Representations*, 2018.

[86] Ł. Maziarka et al., "Mol-CycleGAN: A generative model for molecular optimization," *J. Cheminformatics*, vol. 12, no. 1, pp. 1–18, 2020.

[87] J. You et al., "Graph convolutional policy network for goal-directed molecular graph generation," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 6410–6421.

[88] Z. Zhou et al., "Optimization of molecules via deep reinforcement learning," *Sci. Rep.*, vol. 9, no. 1, pp. 1–10, 2019.

[89] W. Jin, R. Barzilay, and T. Jaakkola, "Composing molecules with multiple property constraints," 2020, arXiv:2002.03244.

[90] B. Chen, L. Sun, and X. Han, "Sequence-to-Action: End-to-End Semantic Graph Generation for Semantic Parsing," in *Proc. 56th Annu. Meeting Assoc. Comput. Linguistics*, 2018, pp. 766–777.

[91] Y. Wang et al., "A neural transition-based approach for semantic dependency graph parsing," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 5561–5568.

[92] S. Yang et al., "Learn to generate time series conditioned graphs with generative adversarial nets," 2020, *arXiv:2003.01436*.

[93] J. Liu, Y. Chi, and C. Zhu, "A dynamic multiagent genetic algorithm for gene regulatory network reconstruction based on fuzzy cognitive maps," *IEEE Trans. Fuzzy Syst.*, vol. 24, no. 2, pp. 419–431, Apr. 2016.

[94] C. Yang et al., "Conditional structure generation through graph variational generative adversarial nets," in *Proc. 33rd Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 1338–1349.

[95] E. Jonas, "Deep imitation learning for molecular inverse problems," in *Proc. 33rd Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 4991–5001.

[96] Y. Li, L. Zhang, and Z. Liu, "Multi-objective de novo drug design with conditional graph generative model," *J. Cheminform.*, vol. 10, no. 1, 2018, Art. no. 33.

[97] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. Int. Conf. Learn. Representations*, 2017.

[98] J. Kawahara et al., "BrainNetCNN: Convolutional neural networks for brain networks; towards predicting neurodevelopment," *NeuroImage*, vol. 146, pp. 1038–1049, 2017.

[99] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 3483–3491.

[100] M. Mirza and S. Osindero, "Conditional generative adversarial nets," 2014, *arXiv:1411.1784*.

[101] S. Amizadeh, S. Matusevych, and M. Weimer, "Learning to solve circuit-sat: An unsupervised differentiable approach," in *Proc. Int. Conf. Learn. Representations*, 2018.

[102] X. Guo, L. Zhao, and C. Nowzari, S. Rafatirad, H. Homayoun, and S. M. Pudukotai Dinakarrao, "Deep multi-attributed graph translation with node-edge co-evolution," in *Proc. IEEE Int. Conf. Data Mining*, 2019, pp. 250–259.

[103] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proc. 3rd Int. Conf. Learn. Representations*, 2015.

[104] T. Lei et al., "Simple recurrent units for highly parallelizable recurrence," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, 2018, pp. 4470–4481.

[105] S. Kullback and R. A. Leibler, "On information and sufficiency," *Ann. Math. Statist.*, vol. 22, no. 1, pp. 79–86, 1951.

[106] A. Gretton et al., "A kernel two-sample test," *J. Mach. Learn. Res.*, vol. 13, no. Mar, pp. 723–773, 2012.

[107] K. Xu et al., "How powerful are graph neural networks?," in *Proc. Int. Conf. Learn. Representations*, 2018.

[108] U. Kang, H. Tong, and J. Sun, "Fast random walk graph kernel," in *Proc. 12th SIAM Int. Conf. Data Mining*, 2012, pp. 828–838.

[109] G. Jurman, R. Visintainer, and M. Filosi, S. Riccadonna, and C. Furlanello, "The HIM glocal metric and kernel for network comparison and classification," in *Proc. IEEE Int. Conf. Data Sci. Adv. Analytics*, 2015, pp. 1–10.

[110] P. Bonacich, "Power and centrality: A family of measures," *Amer. J. Sociol.*, vol. 92, no. 5, pp. 1170–1182, 1987.

[111] L. C. Freeman, "Centrality in social networks conceptual clarification," *Social Netw.*, vol. 1, no. 3, pp. 215–239, 1978.

[112] N. Shervashidze et al., "Weisfeiler-lehman graph kernels," *J. Mach. Learn. Res.*, vol. 12, no. 77, pp. 2539–2561, 2011.

[113] B. Samanta et al., "NeVAE: A deep generative model for molecular graphs," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 1110–1117.

[114] V. Golkov et al., "Protein contact prediction from amino acid co-evolution using convolutional networks for graph-valued images," in *Proc. 30th Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 4222–4230.

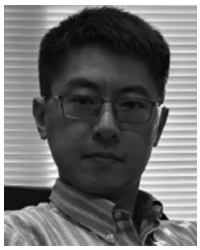
[115] L. Livi et al., "A generative model for protein contact networks," *J. Biomol. Struct. Dyn.*, vol. 34, no. 7, pp. 1441–1454, 2016.

[116] L. Dong and M. Lapata, "Language to logical form with neural attention," in *Proc. 54th Annu. Meeting Assoc. Comput. Linguistics*, 2016, pp. 33–43.

[117] R. Jia and P. Liang, "Data recombination for neural semantic parsing," in *Proc. 54th Annu. Meeting Assoc. Comput. Linguistics*, 2016, pp. 12–22.

[118] C. Maddison and D. Tarlow, "Structured generative models of natural source code," in *Proc. 31st Int. Conf. Mach. Learn.*, 2014, pp. 649–657.
[119] A. T. Nguyen and T. N. Nguyen, "Graph-based statistical language model for code," in *Proc. IEEE 37th Int. Conf. Softw. Eng.*, 2015, pp. 858–868.
[120] M. Allamanis, M. Brockschmidt, and M. Khademi, "Learning to represent programs with graphs," in *Proc. Int. Conf. Learn. Representations*, 2018.
[121] Z. Newsham et al., "Impact of community structure on SAT solver performance," in *Proc. Int. Conf. Theory Appl. Satisfiability Testing*, 2014, pp. 252–268.
[122] J. Giráldez-Cru and J. Levy, "A modularity-based random SAT instances generator," in *Proc. 24th Int. Conf. Artif. Intell.*, 2015, pp. 1952–1958.
[123] J. You et al., "G2SAT: Learning to generate SAT formulas," in *Proc. 33rd Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 10552–10563.
[124] H. Wu and R. Ramanujan, "Learning to generate industrial sat instances," in *Proc. 12th Annu. Symp. Combinatorial Search*, 2019, pp. 206–207.
[125] B. M. Lake et al., "Building machines that learn and think like people," *Behav. Brain Sci.*, vol. 40, 2017, Art. no. e253.
[126] E. Noutahi et al., "Towards interpretable sparse graph representation learning with laplacian pooling," 2019, *arXiv:1905.11577*.
[127] D. Bouchacourt, R. Tomioka, and S. Nowozin, "Multi-level variational autoencoder: Learning disentangled representations from grouped observations," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 2095–2102.
[128] N. Stoehr et al., "Disentangling interpretable generative parameters of random and real-world graphs," 2019, *arXiv:1910.05639*.
[129] X. Guo et al., "Interpretable deep graph generation with node-edge co-disentanglement," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2020, pp. 1697–1707.
[130] Y. Du et al., "Disentangled spatiotemporal graph generative models," 2022, *arXiv:2203.00411*.
[131] X. Guo, Y. Du, and L. Zhao, "Deep generative models for spatial networks," in *Proc. 27th ACM SIGKDD Conf. Knowl. Discov. Data Mining*, 2021, pp. 505–515.
[132] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," 2015, *arXiv:1511.06434*.
[133] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 4401–4410.

Xiaojie Guo received the PhD degree in the information technology from George Mason University, supervised by Dr. Liang Zhao. Her research interests include data mining, artificial intelligence, and machine learning, with special interests in deep learning on graphs, deep graph transformation, deep graph generation as well as disentangled representation learning. She received the Best Paper Award from ICDM, in 2019.



Liang Zhao is an assistant professor with the Department of Computer Science, Emory University. His research interests include data mining, and machine learning, with special interests in spatiotemporal data mining and deep learning on graphs. He has published more than 80 papers in top-tier conferences and journals such as KDD, ICDM, *IEEE Transactions on Knowledge and Data Engineering*, *Proceedings of the IEEE*, *ACM Transactions on Knowledge Discovery from Data*, IJCAI, AAAI, WWW, SIGSPATIAL, and SDM. He has won NSF CAREER Award, Amazon Research Award, and Jeffress Trust Award, in 2019.