Received May 28, 2021, accepted July 1, 2021, date of publication July 19, 2021, date of current version August 5, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3098417

Deep Graph Generators: A Survey

FAEZEH FAEZ[1], YASSAMAN OMMI[2], MAHDIEH SOLEYMANI BAGHSHAH[1], AND HAMID R. RABIEE[1], (Senior Member, IEEE)

[1]Department of Computer Engineering, Sharif University of Technology, Tehran 11155-9517, Iran
[2]Department of Mathematics and Computer Science, Amirkabir University of Technology, Tehran 15914, Iran

Corresponding authors: Hamid R. Rabiee (rabiee@sharif.edu) and Mahdieh Soleymani Baghshah (soleymani@sharif.edu)

ABSTRACT Deep generative models have achieved great success in areas such as image, speech, and natural language processing in the past few years. Thanks to the advances in graph-based deep learning, and in particular graph representation learning, deep graph generation methods have recently emerged with new applications ranging from discovering novel molecular structures to modeling social networks. This paper conducts a comprehensive survey on deep learning-based graph generation approaches and classifies them into five broad categories, namely, autoregressive, autoencoder-based, reinforcement learning-based, adversarial, and flow-based graph generators, providing the readers a detailed description of the methods in each class. We also present publicly available source codes, commonly used datasets, and the most widely utilized evaluation metrics. Finally, we review current trends and suggest future research directions based on the existing challenges.

INDEX TERMS Generative models, deep learning, graph data, deep graph generators, molecular graph generation.

I. INTRODUCTION

Recently, with the rapid development of data collection and storage technologies, an increasing amount of data that needs to be processed is available. In many research areas, including biology, chemistry, pharmacy, social networks, and knowledge graphs, there exist some relationships between data entities that, if taken into account, more valuable features can be extracted, yielding more accurate predictions. Using graph data structure is a common way to represent such data, and therefore graph analysis research has attracted considerable attention.

Over the past few years, machine learning-based graph studies have made significant progress, mainly focusing on graph representation learning [1]–[5] whose main goal is to find appropriate embeddings for nodes, edges, and the whole graph in a continuous low-dimensional space. These embeddings can be further utilized by various downstream tasks such as graph visualization, clustering, node classification, and link prediction [6]. Beyond the prominent field of graph representation learning, graph-related research further includes other areas such as graph matching [7], [8], adversarial attack and defense on graph-based neural networks [9], [10], and graph attention networks [11], [12]. Graph generation is also another research line aiming to generate new graph structures with some desired properties, which dates back to 1960 [13] and is followed by several other approaches [14]–[17]. However, the early methods generally use hand-engineered processes to create graphs with predefined statistical properties and, despite their simplicity, are not capable enough to capture complicated graph dependencies.

Thanks to the recent successes of deep learning techniques and algorithms, deep generative models, which aim to generate novel samples from a similar distribution as the training data, have received a lot of attention in various data domains such as image [18], [19], text [20], [21], and speech [22], [23]. Subsequently, studies related to deep learning-based graph generators have started a little later, which, unlike the traditional approaches, can directly learn from data and eliminate the need for using hand-designed procedures.

In the last few years, deep learning-based graph generation has gradually attracted the attention of many researchers with applications ranging from discovering new molecular structures to modeling social networks. Therefore, due to the expanding development of the modern graph generation approaches, which we refer to as Deep Graph Generators (DGGs), and also considering their applications in various research areas, there is a need for articles that specifically review them.

The associate editor coordinating the review of this manuscript and approving it for publication was Fanbiao Li.

TABLE 1. Categorization, key characteristic, and representative publications among deep graph generators.

Category	Key Characteristic	Publications
Autoregressive DGGs	Adopting a sequential generation strategy, either node-by-node or edge-by-edge	[24]–[49]
Autoencoder-Based DGGs	Making the generation process dependent on latent space variables	[37]–[42], [50]–[62]
RL-Based DGGs	Utilizing reinforcement learning algorithms to induce desired properties in the generated graphs	[26], [43]–[49], [63]
Adversarial DGGs	Employing generative adversarial networks (GANs) [64] to generate graph structures	[43], [45], [61]–[63], [65]–[70]
Flow-Based DGGs	Learning a mapping from the complicated graph distribution into a distribution mostly modeled as a Gaussian for calculating the exact data likelihood	[35], [36], [60], [71]

To address this need, we conduct a comprehensive survey on DGGs that exclusively reviews these methods and their applications. Our key contributions are as follows:

- We first divide the existing approaches into five broad categories: autoregressive DGGs, autoencoder-based DGGs, reinforcement learning-based (RL-based) DGGs, adversarial DGGs, and flow-based DGGs, providing the readers with detailed descriptions of the methods in each category and comparing them from different aspects. This categorization is based on the model architectures, adopted generation strategies, or optimization objectives (the categories may sometimes overlap so that a method can belong to more than one category). Table 1 summarizes the main characteristics of these categories, along with the most prominent approaches belonging to each of them.
- We review the most outstanding applications explored by the current DGGs and discuss their challenges along with the solutions that have yet been proposed to address them. We further suggest some other problems that can potentially be solved by a graph generation perspective as future applications.
- We give the interested readers easy access to additional resources and implementation details. Specifically, we categorize the most important datasets used by DGGs and provide their statistical information along with their accessible links. We also collect open-source codes and summarize the evaluation metrics.
- We discuss current trends of deep graph generators in both techniques and applications and suggest future research directions accordingly.

We believe this article helps readers within the field to classify their current knowledge, get informed of the latest achievements, have quick access to the related resources, and gain a useful view of the field's future. It is also a good source for readers outside the field to get acquainted with the modern graph generation research area and quickly obtain an overview of its recent techniques and trends.

Related works. So far, several surveys have reviewed deep graph-related approaches such as those mainly focusing on graph representation learning methods [6], [72]–[78], graph attention models [79], knowledge graph research [80], [81], attack and defense techniques on graph data [82], and graph matching approaches [83], [84]. Although most of these surveys have made a passing reference to some modern graph generators, this field requires individual attention due to its value and growing popularity. Recently, and somewhat concurrent to our work, Guo and Zhao [85] have exclusively reviewed this research field from totally different categorizations and perspectives than we do. Specifically, they have classified the existing methods based on properties of their generation process (e.g., whether the generation process is conditional or unconditional, one-shot or sequential, and etc.), while we categorize them mainly according to their training objectives and model architectures in five classes (i.e., autoregressive, autoencoder-based, RL-based, adversarial, and flow-based) and identify key characteristics of the approaches in each category. Moreover, the authors of [85] neither provide supplementary resources related to DGGs (i.e., datasets and source codes) nor they investigate trends of techniques and applications. These are both included in this paper with additional analyses of each category's methods and their progression over time. We also review the most recent approaches, some of which are not covered in [85].

The rest of this article is organized as follows. Section II briefly summarizes notations used in this survey and formulates the problem of deep graph generation. Sections III to VII provide a detailed review of the existing DGGs in each of the five categories discussed above. Section VIII provides an overview of the categories' characteristics and compares them from various perspectives. Section IX classifies the current applications and suggest some potential future ones. Section X goes through implementation details by summarizing commonly used datasets, widely utilized evaluation metrics, and available source codes. Section XI reviews current trends in both techniques and applications and discusses future research directions. Finally, Section XII concludes the survey.

II. NOTATIONS AND PROBLEM FORMULATION

This section reviews the notations used in the survey and provides a problem formulation for generating a set of plausible graphs.

TABLE 2. Commonly used notations.

Notations	Descriptions
G	A graph.
V	The node set of a graph.
E	The edge set of a graph.
p(G)	The graph data distribution.
n	The number of nodes, n=|V|.
N	The largest graph size in the dataset.
m	The number of edges, m=|E|.
π	A node ordering for a graph G.
A^{π}	The adjacency matrix corresponding to a graph G under a node ordering π.
S^{π}	The sequence corresponding to nodes of a graph under a node ordering π.
$S^{edge,\ \pi}$	The sequence corresponding to edges of a graph.
$emb(.)$	An embedding function.
$[x, y]$	The concatenation of x and y.
$\mathcal{N}_u$	The neighbor set of node u.

A. NOTATIONS

We represent a graph as $G = (V, E)$, where V is the graph's node set, and E denotes its edge set, with $|V| = n$ and $|E| = m$. N also indicates the largest graph size in the dataset. There are $n!$ possible node orderings for the graph; thus, if we choose an ordering π, the graph can be represented by the corresponding adjacency matrix $A^{\pi} \in \mathbb{R}^{n\times n}$. Moreover, we can represent the graph with sequences of its ordered nodes or edges denoted by $S^{node,\ \pi}$ and $S^{edge,\ \pi}$, respectively, where the former is denoted by the shorter form S^{π} in the following for simplicity. The notations are summarized in Table 2.

B. PROBLEM FORMULATION

Given a dataset of graphs $\mathscr{D}_G$ with the underlying data distribution $p(G)$ (i.e., for each graph G in the dataset, $G \sim p(G)$), a DGG aims to learn how to obtain new samples from the data distribution by employing deep neural networks. Specifically, this can be done by either estimating the $p(G)$ first and then sample from the estimated distribution or acquiring an implicit strategy, which only learns how to sample from the distribution without explicitly modeling it.

III. AUTOREGRESSIVE DEEP GRAPH GENERATORS

In this section, we review those approaches generating graph structures sequentially in a step-wise fashion, where the prediction at each time step is affected by the previous outputs. We further divide them into recurrent and non-recurrent approaches, where the former captures the generation history by employing recurrent units, while the latter makes decisions directly based on the latest partially generated graph. The main characteristics of these methods are summarized in Table 3.

A. RECURRENT DGGs

Recurrent DGGs are a bunch of autoregressive deep graph generators that use RNNs, namely long short-term memory (LSTM) [86] or gated recurrent units (GRU) [87], to exert the influence of the generation history on the current decision. Here, we provide a detailed review of these methods in two subcategories.

1) NODE-BY-NODE GENERATORS

Most of the autoregressive methods append one new node at a time into the already generated graph. For example, Li *et al.* [24] propose to generate molecular graphs sequentially, where the generation process initiates by adding a node to an empty graph. It then continues by iteratively deciding whether to append a new node to the graph, connect the lastly added node to the previous ones, or terminate the process. To this end, the authors propose two architectures, namely MolMP and MolRNN, to determine probabilities for each of these three actions. More precisely, MolMP decides based on the graph's current state, modeling the generation as a Markov Decision Process. It first calculates an initial embedding for graph nodes followed by several convolutional layers and an aggregation operation to obtain a graph-level representation. It then passes both the node-level and graph-level embeddings through MLP and softmax layers to compute the probabilities required for action selection. MolRNN, on the other hand, exploits molecule level recurrent units to make the generation history affect the current decision, which improves the model's performance. It adopts the same approach as MolMP to obtain embeddings and then updates the recurrent units' hidden state as follows:

$$h_i = f_{trans}(h_{i-1}, h_{v^*}, h_{G_{i-1}}), \tag{1}$$

where f_{trans} is implemented using GRUs, h_{v^*} is the latest appended node embedding, and $h_{G_{i-1}}$ denotes the representation for the graph generated before the i-th generation step. Next, the action probabilities are calculated similarly as MolMP, except that MolRNN replaces h_i by the graph-level representation. Moreover, the authors make the conditional graph generation possible by first converting a given requirement to a conditional code and then modifying the graph convolution to include this code.

You *et al.* [25] propose GraphRNN, another deep autoregressive model with a hierarchical architecture consisting of a graph-level RNN and an edge-level RNN which learns to sample $G \sim p(G)$ without explicitly computing $p(G)$. For this purpose, GraphRNN first defines a mapping f_S from graphs to sequences where for a graph G with n nodes under the node ordering π, the mapping is defined as follows:

$$S^{\pi} = f_S(G, \pi) = (S_1^{\pi}, \ldots, S_n^{\pi}), \tag{2}$$

where each element $S_i^{\pi} \in \{0, 1\}^{i-1}, i \in \{1, \ldots, n\}$ represents the edges between node $\pi(v_i)$ and the previous nodes.

TABLE 3. The main characteristics of autoregressive deep graph generators.

Method	Recurrent	Generation Strategy	Attention Mechanism	Features	Conditional Generation
MolMP [24]	No	Node-by-node	No	Node/Edge	Yes
MolRNN [24]	Yes	Node-by-node	No	Node/Edge	Yes
GraphRNN [25]	Yes	Node-by-node	No	-	No
MolecularRNN [26]	Yes	Node-by-node	No	Node/Edge	No
Bacciu et al. [27], [28]	Yes	Edge-by-edge	No	-	No
GraphGen [29]	Yes	Edge-by-edge	No	Node/Edge	No
GRAN [30]	No	Block of nodes	Yes	-	No
GRAM [31]	No	Node-by-node	Yes	Node/Edge	No
AGE [32]	No	Node-by-node	Yes	Node	Yes
DeepGMG [33]	No	Node-by-node	Yes	Node/Edge	Yes
BiGG [34]	No	Node-by-node	No	-	No

Since for undirected graphs, there exists the mapping function $f_G(S^\pi) = G$, it is possible to sample G at inference time by first sampling $S^\pi \sim p(S^\pi)$ and then applying f_G, which obviates the need to compute $p(G)$ explicitly. To learn $p(S^\pi)$, due to the sequential nature of S^π, $p(S^\pi)$ can be further decomposed as in Eq. (3), which is modeled by an RNN with state transition and output functions defined in Eq. (4) and (5), respectively:

$$p(S^\pi) = \prod_{i=1}^{n+1} p(S_i^\pi | S_1^\pi, \ldots, S_{i-1}^\pi) = \prod_{i=1}^{n+1} p(S_i^\pi | S_{<i}^\pi), \quad (3)$$

$$h_i^{node} = f_{trans}^{node}(h_{i-1}^{node}, \mathrm{In}_i^{node}), \quad \mathrm{In}_i^{node} = S_{i-1}^\pi, \quad (4)$$

$$\theta_i = f_{out}(h_i^{node}), \quad (5)$$

where f_{trans}^{node} is implemented using a GRU and serves as the graph-level RNN that maintains the state of the graph generated so far. Furthermore, the authors propose two varients for the implementation of f_{out}. First, they propose GraphRNN-S, a simple variant that does not consider dependencies between edges and models $p(S_i^\pi | S_{<i}^\pi)$ as a multivariate Bernoulli distribution. Next, to fully capture the complex edge dependencies, they propose the full GraphRNN model as illustrated in Figure 1, which approximates f_{out} by another RNN (i.e., the edge-level RNN) formulated as follows:

$$h_{i,j}^{edge} = f_{trans}^{edge}(h_{i,j-1}^{edge}, \mathrm{In}_j^{edge}), \quad \mathrm{In}_j^{edge} = S_{i,j-1}^\pi, \quad h_{i,0}^{edge} = h_i^{node}. \quad (6)$$

Furthermore, GraphRNN introduces a BFS node ordering scheme to improve scalability with two benefits. First, it will suffice to train the model on all possible BFS orderings, rather than all possible node permutations. Second, it reduces the number of edges to be predicted in the edge-level RNN.

Subsequently, several graph generation methods have been proposed inspired by GraphRNN. For example, Liu *et al.* [88] propose two further variants for the implementation of f_{out} function in Eq. (5), namely RNN-Transf and GraphRNN+Attn. Specifically, RNN-Transf replaces the edge-level RNN in the full GraphRNN model with a vanilla Transformer [89] decoder consisting of a self-attention sublayer and a graph-state attention sublayer with the memory

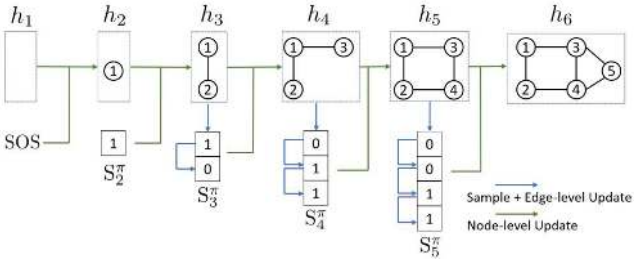

FIGURE 1. An illustration of the graph generation procedure at inference time proposed in GraphRNN [25] (reprinted with permission). Green arrows denote the graph-level RNN, and blue arrows represent the edge-level RNN.

from hidden states of the node-level RNN. GraphRNN+Attn, on the other hand, maintains the edge-level RNN. More precisely, it is an additive attention mechanism in the edge-level RNN that computes the attention weights in each step using the last hidden states of the node-level RNN as well as the current hidden state of the edge-level RNN.

MolecularRNN [26] extends GraphRNN to generate realistic molecular graphs with desired chemical properties. As in molecular graphs, both nodes and edges have types, likelihood formulation in Equation (3) is rewritten as follows:

$$p(S^\pi, C^\pi) = \prod_{i=1}^{n+1} p(C_i^\pi | S_{<i}^\pi, C_{<i}^\pi) p(S_i^\pi | C_i^\pi, S_{<i}^\pi, C_{<i}^\pi), \quad (7)$$

where $S_{i,j}^\pi \in \{0, 1, 2, 3\}$ is the categorical edge type that corresponds to no, single, double, or triple bonds, and $C_i^\pi \in \{1, 2, \ldots, K\}$ determines node (atom) type. Then, MolecularRNN substitutes the graph-level RNN input in Eq. (4) with the embeddings of categorical inputs as in Eq. (8):

$$\mathrm{In}_i^{node} = [emb(S_{i-1}^\pi), emb(C_{i-1}^\pi)]. \quad (8)$$

Furthermore, a two-layer MLP with softmax output activation is added on top of the hidden states of both graph-level and edge-level RNNs to predict node and edge types, respectively. After likelihood pretraining on the molecular datasets, the model is fine-tuned with the policy gradient algorithm to shift the distribution of the generated samples to some desired

chemical properties, namely, lipophilicity, drug-likeness, and melting point. Thus, the MolecularRNN acts as a policy network to output the probability of the next action given the current state, where the set of states consists of all possible sub-graphs and the possible atom connections to the existing graph, for all the atom types, serve as the action set. Moreover, each valid molecule is considered as a final state s_n, where its corresponding final reward is denoted by $r(s_n)$. The intermediate rewards $r(s_i), 0 < i < n$ are also obtained by discounting $r(s_n)$ as in the following loss function formula:

$$\mathscr{L}(\theta) = -\sum_{i=1}^{n} r(s_n) \cdot \gamma^i \cdot \log p(s_i|s_{i-1};\theta), \tag{9}$$

where γ is the discount factor and the transition probabilities $p(s_i|s_{i-1};\theta)$ are the elements of the product in Eq(7). Furthermore, MolecularRNN introduces the structural penalty for atoms violating valency constraints during training. It also adopts a valency-based rejection sampling method during inference, which guarantees the generated samples' validity.

Sun and Li [90] learn a mapping from a source to a target graph by adopting an encoder-decoder based approach, where the encoder utilizes recurrent based models to encode the source graph and the decoder generates the target graph in a node-by-node fashion, which makes it necessary to consider an ordering over nodes. Therefore, the authors first introduce a procedure to transform a graph G into a DAG (Directed Acyclic Graph) to provide the required node ordering. They then obtain embeddings for each of the DAG's nodes by proposing two encoders: an Energy-Flow encoder and a Topology-Flow encoder, where the former utilizes only the information of adjacent nodes, while the latter exploits both the adjacent and non-adjacent nodes' information. Afterward, the decoder sequentially generates the target graph conditioned on the source graph by adopting a relatively similar generation strategy as the GraphRNN.

So far, we have studied GraphRNN [25] as one of the most widely used deep graph generators and then reviewed the subsequent graph generation approaches inspired by it; each generates different types of graphs from general to molecular ones. Moreover, there are also other methods that use GraphRNN as a basis for solving some application-specific problems. For example, REIN [91] proposes to autoregressively generate meshes from input point clouds inspired by GraphRNN so that in each generation step, it predicts edges from the newly introduced point to all the previous ones. The generated mesh can then be used for the task of 3D object reconstruction. DeepNC [92] is another GraphRNN-based approach that proposes a network completion algorithm to infer the missing parts of a network. Specifically, it first trains GraphRNN to learn a likelihood over the data distribution. The method then formulates an optimization problem to infer the missing parts of a partially observed input graph in such a way that maximizes the learned likelihood.

2) EDGE-BY-EDGE GENERATORS

In addition to the methods discussed so far, there also exist other approaches adopting an edge-based generation strategy. Bacciu *et al.* [28] propose to generate a sequence of edges for each graph instead of generating graphs node-by-node. They first convert a graph G under the node ordering π to an ordered edge sequence $S^{edge,\,\pi} = [S_1^{edge,\,\pi}, \ldots, S_m^{edge,\,\pi}]$, where $S_i^{edge,\,\pi} = (u_i^\pi, v_i^\pi)$ is the i-th edge in the sequence that connects the source node u_i^π to the destination node v_i^π (here u_i^π and v_i^π are IDs assigend to graph nodes by π). Note that the sequence is ordered, that is $S_i^{edge,\,\pi} \leq S_{i+1}^{edge,\,\pi}$ iff $u_i^\pi < u_{i+1}^\pi$ or $(u_i^\pi = u_{i+1}^\pi$ and $v_i^\pi < v_{i+1}^\pi)$. Then, the authors define $U^\pi = [u_1^\pi, \ldots, u_m^\pi]$ and $V^\pi = [v_1^\pi, \ldots, v_m^\pi]$ as sequences of the source and destination node IDs, respectively and decompose the edge sequence probability as follows:

$$S^{edge,\,\pi} = p(U^\pi)p(V^\pi|U^\pi), \tag{10}$$

where $p(U^\pi)$ and $p(V^\pi|U^\pi)$ are approximated with two RNNs. Specifically, `RNN1` is used to estimate $p(U^\pi)$ with the following transition and output functions:

$$\begin{aligned} &h_i^{\texttt{RNN1}} = f_{trans}(h_{i-1}^{\texttt{RNN1}}, \text{In}_i^{\texttt{RNN1}}), \quad \text{In}_i^{\texttt{RNN1}} = emb(u_{i-1}^\pi) \\ &p(u_i^\pi|u_{i-1}^\pi, h_{i-1}^{\texttt{RNN1}}) \\ &\qquad = f_{out}(h_i^{\texttt{RNN1}}) = \text{Softmax}(Lin(h_i^{\texttt{RNN1}})), \end{aligned} \tag{11}$$

where f_{trans} is implemented as a GRU and Lin is a linear projection to map the recurrent output to the node ID space. Once all of the U^π's elements are generated, the last recurrent state of `RNN1` is used to initialize the state of `RNN2`, and the process moves to `RNN2` that is given U^π as input. Thus `RNN2` computes the probability distribution of $p(V^\pi|U^\pi)$ with the same architecture as `RNN1` by approximating $p(v_i|u_i, h_{i-1}^{\texttt{RNN2}})$ each step.

Similarly, GraphGen [29] proposes another edge-based generation strategy that adds a single edge to the already generated graph at each stage. To this end, the method first converts a graph G to a sequence $S^{edge} = [S_1^{edge}, \ldots, S_m^{edge}]$ using the minimum DFS code [93], where each S_i^{edge} corresponds to an edge $e = (u, v)$ and is described using a 5-tuple $(t_u, t_v, L_u, L_e, L_v)$, where t_u is the timestamp assigned to node u during the DFS traversal, and L_u and L_e denote the node and edge labels, respectively. As the minimum DFS codes are canonical labels, and thus there is a one-to-one mapping between a graph and its corresponding sequence, there is no longer need to deal with multiple representations for the same graph under different node permutations during training, which improves the method scalability. Then, GraphGen takes a similar approach to GraphRNN [25] to decompose $p(S^{edge})$ as follows:

$$p(S^{edge}) = \prod_{i=1}^{m+1} p(S_i^{edge}|S_1^{edge}, \ldots, S_{i-1}^{edge}) = \prod_{i=1}^{m+1} p(S_i^{edge}|S_{<i}^{edge}), \tag{12}$$

where m is the number of edges, and making the simplifying assumption that t_u, t_v, L_u, L_e, and L_v are independent,

reduces $p(S_i^{edge}|S_{<i}^{edge})$ in Eq. (12) to:

$$\begin{aligned} p(S_i^{edge}|S_{<i}^{edge}) &= p((t_u, t_v, L_u, L_e, L_v)|S_{<i}^{edge}) \\ &= p(t_u|S_{<i}^{edge}) \times p(t_v|S_{<i}^{edge}) \times p(L_u|S_{<i}^{edge}) \\ &\quad \times p(L_e|S_{<i}^{edge}) \times p(L_v|S_{<i}^{edge}). \end{aligned} \tag{13}$$

To capture conditional distributions in Eq. (13), the authors propose to use a custom LSTM with the transition function f_{trans} in Eq. (14), and five separate output functions for each component of the 5-tuple. For example, f_{t_u} in Eq. (15) is utilized for predicting t_u, where $\sim_M$ represents sampling from a multinomial distribution.

$$h_i = f_{trans}(h_{i-1}, \text{In}_i), \quad \text{In}_i = emb(S_{i-1}^{edge}), \tag{14}$$

$$t_u \sim_M \ \theta_{t_u} = f_{t_u}(h_i), \tag{15}$$

$$S_i^{edge} = concat(t_u, t_v, L_u, L_e, L_v). \tag{16}$$

Figure 2 outlines the proposed pipeline.

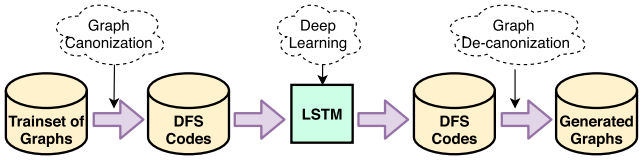

FIGURE 2. Flowchart of GraphGen [29].

B. NON-RECURRENT DGGs

There are some other autoregressive methods that, unlike the recurrent models, do not consider the entire history, and instead, they only focus on the latest version of the partially generated graph at each time step. To better review these methods, we further divide them into two following subsections.

1) ATTENTION-BASED METHODS

Here, we review the methods in which the attention mechanism plays a key role. In this regard, GRAN [30] proposes to generate one block of nodes and associated edges at each generation step by optimizing the following likelihood:

$$p(L^\pi) = \prod_{t=1}^{T} p(L_{\mathbf{b_t}}^\pi | L_{\mathbf{b_1}}^\pi, \ldots, L_{\mathbf{b_{t-1}}}^\pi), \tag{17}$$

where L^π is the lower triangular part of the adjacency matrix A^π, B denotes the block size, $\mathbf{b_t} = \{B(t-1)+1, \ldots, Bt\}$ is the set of row indices for the t-th block of L^π, and $T = \lceil \frac{N}{B} \rceil$ is the number of graph generation steps. For the t-th step, GRAN adds B new nodes to the already-generated subgraph and connects them with each other as well as the previous $B(t-1)$ nodes to acquire an augmented graph as depicted in Figure 3. The authors then apply the following graph neural network with attentive messages on the augmented graph to get updated node representations:

$$\begin{aligned} m_{ij}^r &= f(h_i^r - h_j^r), \quad \tilde{h}_i^r = [h_i^r, x_i], \quad a_{ij}^r = \sigma\big(g(\tilde{h}_i^r - \tilde{h}_j^r)\big) \\ h_i^{r+1} &= \text{GRU}(h_i^r, \sum_{j \in \mathcal{N}(i)} a_{ij}^r m_{ij}^r), \end{aligned} \tag{18}$$

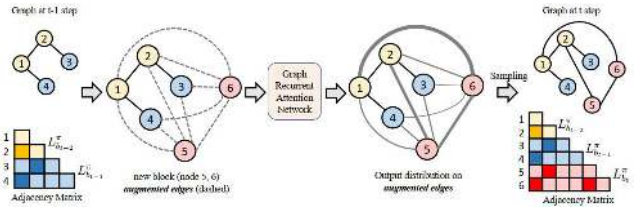

FIGURE 3. An overview of GRAN [30] (reprinted with permission). Dashed lines are augmented edges. Nodes with the same color belong to the same block (block size = 2).

where h_i^r is the representation for node i after round r, m_{ij}^r is the message vector from node i to j, x_i indicates whether node i is in the previously generated nodes or the newly added ones, and a_{ij}^r is an attention weight associated with $edge(i, j)$. Both the message function f and the attention function g are implemented as 2-layer MLPs with ReLU nonlinearities. After R rounds of message passing, the final node representation vectors h_i^R for each node i is obtained, and then GRAN models the conditional probability in Eq. (17) with a mixture of Bernoulli distributions to capture edge dependencies via K latent mixture components:

$$\begin{aligned} p(L_{\mathbf{b_t}}^\pi | L_{\mathbf{b_1}}^\pi, \ldots, L_{\mathbf{b_{t-1}}}^\pi) &= \sum_{k=1}^{K} \alpha_k \prod_{i \in \mathbf{b_t}} \prod_{1 \le j \le i} \theta_{k,i,j} \\ \alpha_1, \ldots, \alpha_K &= \text{Softmax}\Big(\sum_{i \in \mathbf{b_t}, 1 \le j \le i} \text{MLP}_\alpha(h_i^R - h_j^R) \Big) \\ \theta_{1,i,j}, \ldots, \theta_{K,i,j} &= \sigma\big(\text{MLP}_\theta(h_i^R - h_j^R)\big). \end{aligned} \tag{19}$$

GRAM [31] proposes to combine graph convolutional networks with graph attention mechanisms to obtain richer features during the graph generation process, where the proposed graph attention mechanism extends the one in [89] by introducing bias terms as a function of the shortest path between nodes. In particular, GRAM tries to maximize the following likelihood, which is somewhat similar to Eq. (7):

$$\begin{aligned} &p(A^\pi, C^\pi) \\ &= \prod_{i=1}^{n+1} p(C_i^\pi | A_{<i,<i}^\pi, C_{<i}^\pi) \prod_{j=1}^{i-1} p(A_{j,i}^\pi | A_{<j,i}^\pi, C_i^\pi, A_{<i,<i}^\pi, C_{<i}^\pi). \end{aligned} \tag{20}$$

To this end, the authors propose an architecture that consists of three networks, namely, feature extractor, node estimator, and edge estimator. Firstly, the feature extractor extracts the local and global information using graph convolution layers and graph attention layers, respectively, where an attention layer employs a self-attention mechanism with the query, key, and value that are set to the node feature vectors. A graph pooling layer then aggregates all node features into a graph feature vector, denoted as h^G, by summing them up. Next, the node estimator determines a label for the new node based on the feature vector of the graph generated so far. Thereafter, the edge estimator predicts labels for edges between the newly added node and those already exist in

the graph one after the other using a source-target attention mechanism as follows:

$$A^{\pi}_{j,i} = \text{Softmax}(g_{EE}(h^v_j, h^G, h^v_i, h^e_{<j})), \tag{21}$$

where g_{EE} is a three-layer feedforward network, h^v_j and h^v_i are label embeddings of node v_j and the new node v_i, respectively, and $h^e_{<j}$ is computed using a source-target attention with $\text{Concat}(h^v_j, h^v_i)$ as its query and $\{\text{Concat}(h^v_t, h^v_i, h^e_{t,i})|t = 1, \ldots, j-1\}$ as both the key and value.

AGE [32] introduces another attention-based generative model, which is conditioned on some input graphs. In other words, the method takes an existing source graph as input and generates a transformed version of it, modeling its evolution. To this end, the authors propose an encoder-decoder based architecture, where its decoder autoregressively generates the target graph in a node-by-node fashion. More specifically, the encoder first applies the self-attention mechanism to the source graph in order to learn its nodes' representations. Then, at each generation step, the decoder first adopts a similar self-attention mechanism as the encoder, followed by source-target attention, which discovers the correlations between the nodes in the source graph and the ones in the already generated target graph. This way the decoder computes a representation for the graph generated so far, which will be further used to predict the new node's label and connections.

2) OTHER METHODS

Besides the attention-based methods reviewed above, other autoregressive non-recurrent DGGs either do not use attention at all, or the attention mechanism does not play a decisive role in their generation process. For example, DeepGMG [33] proposes a sequential graph generation process which can be seen as the following sequence of decisions: (1) whether to add a new node of a particular type or not (with probabilities provided by the $f_{addnode}$ in Eq. (22), where h_G is the graph representation vector, and f_{an} is an MLP that maps h_G to the action output space), if a node type is selected (2) the model decides whether to continue connecting the newly added node to the existing graph or not (referring to Eq. (23), where $h^{(T)}_v$ is embedding of the new node v after T rounds of propagation in a graph neural network, and f_{ae} is another MLP), if yes (3) it selects a node already in the graph and connects it to the new node (referring to the Eq. (24), where f_s maps pairs $h^{(T)}_u$ and $h^{(T)}_v$ to a score s_u). The algorithm goes back to step (2) and repeats until the model decides not to add another edge. Finally, the algorithm goes back to step (1) to add subsequent nodes or terminate the process.

$$f_{addnode}(G) = \text{Softmax}\,(f_{an}(h_G)), \tag{22}$$

$$f_{addedge}(G, v) = \sigma(f_{ae}(h_G, h^{(T)}_v)), \tag{23}$$

$$s_u = f_s(h^{(T)}_u, h^{(T)}_v), \quad \forall u \in V$$

$$f_{nodes}(G, v) = \text{Softmax}\,(s). \tag{24}$$

DeepGG [94] further extends DeepGMG [33] by adding the idea of finite state machines into the generation process. Furthermore, similar to the GraphRNN [25], DeepGG learns the graph distribution from a sequence called construction sequence, which consists of graph evolutionary actions such as node addition, edge addition, and node deletion.

Recently, BiGG [34] proposes an autoregressive model to increase scalability for generating sparse graphs. To learn a generative model, BiGG uses a single canonical ordering $\pi(G)$ to model each graph G, as in [33], aiming to learn a lower bound on p(G):

$$p(G) = p(V)P(E|V) = p(|V| = n)\sum_{\pi} p(A^{\pi})$$

$$\approx p(|V| = n)p(A^{\pi(G)}), \tag{25}$$

where $p(|V| = n)$ can be directly estimated using an empirical distribution over the graph size. Therefore the goal is only to model $p(A^{\pi(G)})$ under a default canonical ordering, which will be denoted by $p(A)$ in the following. Considering that most real-world graphs are sparse, BiGG generates only the non-zero entries in A in a row-wise manner to enhance efficiency and scalability; thus, the method adopts a recursive strategy inspired by R-MAT [95], for generating each edge as illustrated in the left half of Figure 4. To further improve efficiency, the authors propose to jointly generate all the connections of an arbitrary node u (non-zero entries in the u-th row of A) by autoregressively generating an *edge-binary tree*, as shown in the right half of Figure 4. Finally, BiGG introduces the full autoregressive model that generates the entire adjacency matrix row by row. The full model utilizes the autoregressive models as building blocks:

$$p(A) = p(\{\mathcal{N}_u\}_{u \in V}) = \prod_{u \in V} p(\mathcal{N}_u | \{\mathcal{N}_{u'} : u' < u\}), \tag{26}$$

where $\mathcal{N}_u$ denotes the set of neighbors for node u. More specifically, inspired by Fenwick tree [96], the authors propose a data structure called *row-binary forest* to encode all the *edge-binary trees* generated so far, which will be used to generate new *edge-binary tree* for the current step.

IV. AUTOENCODER-BASED DEEP GRAPH GENERATORS

This section reviews those approaches that employ whether autoencoders (AEs) or VAEs [97] to generate graph structures. In particular, a common practice in these methods is first to encode an input graph into a latent space using GNN [98], GCN [1], or their variants and then start to generate the graph from this latent space embedding. We divide the existing approaches based on their generation granularity level (i.e., adopting an all-at-once generation strategy, using valid substructures as building blocks, or generating graphs in a node-by-node fashion) into the following three subsections. The main characteristics of the most prominent autoencoder-based graph generators are presented in Table 4.

A. ONE-SHOT GENERATORS

A series of autoencoder-based DGGs generate the entire graph all at once. VGAE [50] proposes a graph generation model that primarily aims to perform unsupervised learning

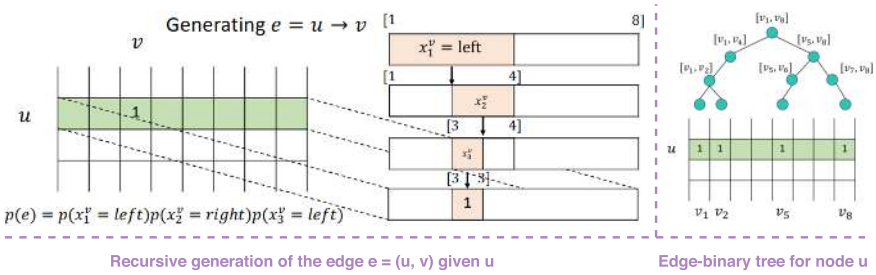

FIGURE 4. An overview of the edge generation procedure in [34] (reprinted with permission).

TABLE 4. The main characteristics of autoencoder-based deep graph generators.

Method	Type	Input	Generation Strategy	Attention Mechanism	Features	Conditional Generation
VGAE [50]	AE/VAE	one single graph	All at Once	No	Node	No
GraphVAE [51]	VAE	dataset of graphs	All at Once	No	Node/Edge	Yes
MPGVAE [52]	VAE	dataset of graphs	All at Once	Yes	Node/Edge	Yes
RGVAE [53]	VAE	dataset of graphs	All at Once	No	Node/Edge	No
Graphite [54]	AE/VAE	one single graph	All at Once	No	Node	No
NED-VAE [55]	β-VAE	dataset of graphs	All at Once	No	Node/Edge	No
DGVAE [56]	VAE	dataset of graphs	All at Once	No	Node	No
JT-VAE [57]	VAE	dataset of graphs	Substructure-Based	No	Node/Edge	No
HierVAE [37]	VAE	dataset of graphs	Substructure-Based	Yes	Node/Edge	Yes
MHG-VAE [58]	VAE	dataset of graphs	Substructure-Based	No	Node/Edge	No
MoleculeChef [38]	WAE	dataset of graphs	Substructure-Based	No	Node/Edge	No
CGVAE [39]	VAE	dataset of graphs	Node-by-Node	No	Node/Edge	No
DEFactor [40]	AE	dataset of graphs	Node-by-Node	No	Node/Edge	Yes
NeVAE [59]	VAE	dataset of graphs	Node-by-Node	No	Node/Edge	No
GraphVRNN [41]	VAE	dataset of graphs	Node-by-Node	No	Node	No
Lim et al. [42]	VAE	dataset of graphs	Node-by-Node	No	Node/Edge	Yes

on graphs based on the variational autoencoder [97]. Given a graph G with adjacency matrix A and node feature matrix X, VGAE infers the latent matrix $\mathbf{Z}$ by a two-layer GCN [1]:

$$q(\mathbf{Z}|X,A) = \prod_{i=1}^{n} q(\mathbf{z}_i|X,A), \quad \text{with } q(\mathbf{z}_i|X,A) = \mathcal{N}(\mathbf{z}_i|\mu_i, \text{diag}(\sigma_i^2)), \tag{27}$$

where $\mu = \text{GCN}_\mu(X,A)$ is the matrix of mean vectors μ_i; similarly $\log\sigma = \text{GCN}_\sigma(X,A)$. Then, the generative model is designed as a simple inner product of latent variables as follows:

$$p(A|\mathbf{Z}) = \prod_{i=1}^{n}\prod_{j=1}^{n} p(A_{ij}|\mathbf{z}_i,\mathbf{z}_j), \text{ with } p(A_{ij}=1|\mathbf{z}_i,\mathbf{z}_j) = \sigma(\mathbf{z}_i^T\mathbf{z}_j), \tag{28}$$

where $\sigma(.)$ is the logistic sigmoid function. The model parameters are then learned by optimizing the VAE objective. However, the authors also proposed a more straightforward, non-probabilistic, and autoencoder-based version of the method called GAE. The main limitation of VGAE is that it can only learn from a single input graph. GraphVAE [51], on the other hand, proposes another VAE-based generative model that learns from a dataset of graphs. The method first embeds the input graph into continuous representation $\mathbf{z}$ using a graph convolution network [99] as the encoder $q_\phi(\mathbf{z}|G)$, where the dimensionality of $\mathbf{z}$ is relatively small in order to learn a high-level compression of the input data. Then, the decoder outputs a probabilistic fully-connected graph with a fairly small predefined maximum size directly at once denoted by $\tilde{G}$. The whole GraphVAE model is trained by minimizing the upper bound on negative log-likelihood as follows:

$$\mathscr{L}_{\theta,\phi}(G) = \mathbb{E}_{q_\phi(\mathbf{z}|G)}[-\log p_\theta(G|\mathbf{z})] + KL[q_\phi(\mathbf{z}|G)||p(\mathbf{z})], \tag{29}$$

where $q_\phi(\mathbf{z}|G)$ and $p_\theta(G|\mathbf{z})$ are the encoder posterior and the decoder generative distributions respectively, and ϕ and θ are parameters to be learned. Since no particular ordering of nodes is imposed in neither G nor $\tilde{G}$, the authors further adopt an approximate graph matching algorithm for aligning G with $\tilde{G}$ in order to compute the likelihood $p_\theta(G|\mathbf{z})$ in Eq. (29). However, the growth of GPU memory requirements, number of parameters, and graph matching complexity for

larger graph sizes limit the applicability of GraphVAE only to generate smaller graphs.

MPGVAE [52] further improves GraphVAE by building a message passing neural network (MPNN) into the encoder and decoder of a VAE, eliminating the need for complex graph matching algorithms. In particular, the method first encodes a molecular graph using a variant of MPNNs [100] combined with a graph attention [11] to aggregate the information over each node's neighbors. The encoder then obtains a graph-level representation using the set2set model [101] and uses this representation to parametrize the posterior distribution $q_\phi(\mathbf{z}|G)$. Next, the decoder samples $\mathbf{z} \sim q_\phi(\mathbf{z}|G)$ and projects it to a high dimensional space consisting of several vectors and then passes these vectors through an RNN to compute initial states for the graph nodes. Afterward, it uses an identical MPNN as the encoder to obtain the final representation for each edge and node. The decoder then reconstructs the graph by predicting the atom types and bond types based on these final representations.

RGVAE [53] regularizes the framework of variational autoencoders to generate semantically valid graphs. To impose validity constraints in the training of VAEs, RGVAE transforms a constrained optimization problem to a regularized, unconstrained one by adding inequality constraints to the objective function of VAEs, which forms a Lagrangian function. More precisely, RGVAE minimizes the following loss function in each parameter update:

$$\mathscr{L} = \mathscr{L}_{\theta,\phi}(G) + \mu \sum_i g_i(\theta, \mathbf{z})_+, \quad \text{where } \mathbf{z} \sim p_\theta(\mathbf{z}), \tag{30}$$

where $g_i(\theta, \mathbf{z}) \leq 0$ denotes the i-th validity constraint, $g_+ = max(g, 0)$, and $\mathscr{L}_{\theta,\phi}(G)$ is the standard VAE loss function as in Eq. (29). The training of RGVAE is illustrated in Figure 5, where l denotes the index of a training example and $\underline{l}$ denotes a synthetic example, utilized in the regularization term.

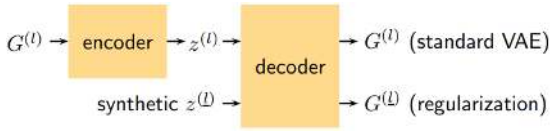

FIGURE 5. The framework of RGVAE [53] (reprinted with permission). The top flow corresponds to the standard VAE, while the bottom flow denotes the regularization, where a synthetic $\mathbf{z}^{(\underline{l})}$ is decoded to compute the constraints $g_i(\theta, \mathbf{z}^{(\underline{l})})_+$.

Graphite [54] proposes a latent variable generative model based on VAE for unsupervised representation learning in large graphs. The method only models graph structure, and any supplementary information such as node features $X \in \mathbb{R}^{n\times k}$ is considered as conditioning evidence. To learn the model parameters θ, Graphite maximizes a lower bound on log-likelihood of the observed adjacency matrix conditioned on X:

$$\log p_\theta(A|X) \geq \mathbb{E}_{q_\phi(\mathbf{Z}|A,X)}\Big[\log \frac{p_\theta(A, \mathbf{Z}|X)}{q_\phi(\mathbf{Z}|A, X)}\Big]. \tag{31}$$

In more detail, the authors take an encoding approach based on the mean-field approximation, which represents graph nodes in the latent space using a graph neural network. Next, they propose an iterative two-step approach as the decoding part: it first constructs an intermediate weighted graph $\hat{A}$ from the latent matrix $\mathbf{Z}$. Then, a parameterized graph neural network updates the latent matrix $\mathbf{Z}^*$. The process alternates between these two steps to refine the graph gradually. More formally, given $\mathbf{Z}$ and X, Graphite iterates over the following two operations:

$$\hat{A} = \frac{\mathbf{Z}\mathbf{Z}^\top}{||\mathbf{Z}||^2} + \mathbf{1}\mathbf{1}^\top, \quad \mathbf{Z}^* = \text{GNN}_\theta(\hat{A}, [\mathbf{Z}, X]), \tag{32}$$

where an additional constant of $\mathbf{1}$ is added into the first operation to ensure entries are non-negative. Finally, it should be noted that similar to VGAE [50], Graphite is also limited to learning from a single input graph.

In addition to the aforementioned graph generative approaches, some initial steps have taken towards making DGGs interpretable. Stoehr *et al.* [102] propose to learn disentangled, interpretable latent variables corresponding to generative parameters of graphs. The main goal of learning such disentangled variables is to make the latent space more interpretable as each latent variable encodes one and only one data property. Therefore, the authors use a GCN as the encoder combined with a deconvolutional neural network as the decoder to minimize the loss function of β-VAE [103]. In this setting, a higher value of β yields the more orthogonalized latent space. To further enforce disentanglement of latent variables, the model also learns an additional *parameter decoder* h, which maps latent variables to generative parameters as illustrated in Figure 6. Recently, NED-VAE [55] proposes a more generalized generative approach for disentanglement learning on attributed graphs that uncovers the independent latent factors in both edges and nodes. In particular, NED-VAE aims to develop a model that can learn the joint distribution of the graph G and three groups of generative independent latent variables, namely, $\mathbf{z}_f$, $\mathbf{z}_e$, and $\mathbf{z}_g$ each of which controls the properties of only nodes, only edges, and the joint patterns between them, respectively. Therefore, inspired by the β-VAE [103] formulation and considering the independence resulted from the disentanglement assumption, the goal is to maximize the following objective function:

$$\begin{aligned}
&\mathscr{L}(\theta, \phi, G, \mathbf{Z}, \beta) \\
&\quad = \mathbb{E}_{q_\phi(\mathbf{Z}|G)}[\log p_\theta(F|\mathbf{z}_f, \mathbf{z}_g) p_\theta(E|\mathbf{z}_e, \mathbf{z}_g)] \\
&\qquad - \beta D_{KL}(q_\phi(\mathbf{z}_f|F)||p(\mathbf{z}_f)) - \beta D_{KL}(q_\phi(\mathbf{z}_e|E)||p(\mathbf{z}_e)) \\
&\qquad - \beta D_{KL}(q_\phi(\mathbf{z}_g|E, F)||p(\mathbf{z}_g)),
\end{aligned} \tag{33}$$

where $E \in \mathbb{R}^{n\times n\times d}$ is the edge attributes tensor, and $F \in \mathbb{R}^{n\times k}$ refers to the node attribute matrix. Based on the above objective, the authors propose an architecture consisting of three sub-encoders, namely, a node encoder, an edge encoder, and a node-edge co-encoder to model the distributions $q_\phi(\mathbf{z}_f|F)$, $q_\phi(\mathbf{z}_e|E)$, and $q_\phi(\mathbf{z}_g|E, F)$, respectively, as depicted in Figure 7. The architecture also consists of two

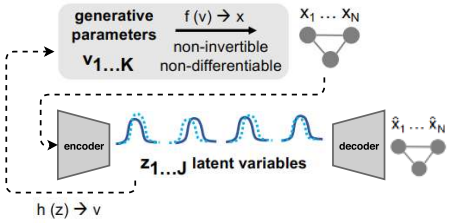

FIGURE 6. Architecture Overview of [102] (reprinted with permission).

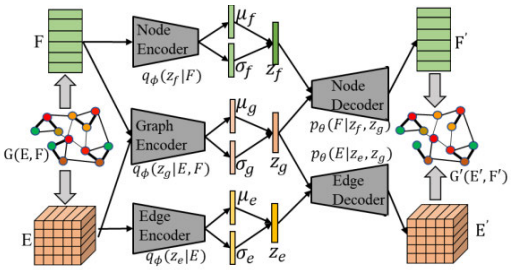

FIGURE 7. The architecture of NED-VAE [55] consisting of three sub-encoders, as well as two sub-decoders (reprinted with permission).

sub-decoders: a node decoder, and an edge decoder to model $p_\theta(F|\mathbf{z}_f, \mathbf{z}_g)$ and $p_\theta(E|\mathbf{z}_e, \mathbf{z}_g)$, respectively. The authors further propose multiple variant models to address different issues, including group-wise disentanglement, variable-wise disentanglement, and the trade-off between reconstruction error and disentanglement performance.

More recently, DGVAE [56] proposes to replace the commonly used Gaussian distribution in VAE-based graph generation models by the Dirichlet distribution as a prior for the latent variables, causing them to describe the graph cluster memberships, which as a result adds interpretability to the model. For this purpose, the authors adopt the same formulation as VGAE [50] in Eq. (27) for the encoding process except that they utilize a GNN variant, named Heatts, proposed by their own and employ the Laplace approximation [104] to model both $q(\mathbf{z}_i|X, A)$ and $p(\mathbf{z}_i)$ as Dirichlet distributions. Furthermore, DGVAE adopts a somewhat similar decoding strategy as VGAE [50] and proves that maximizing the reconstruction term of the model is equivalent to minimizing *balanced graph cut*, which further gives the authors the motivation for designing the Heatts.

B. SUBSTRUCTURE-BASED GENERATORS

There exists a number of works using valid chemical substructures as building blocks to generate more plausible molecular graphs. JT-VAE [57] adopts such a strategy by extending the variational autoencoder framework, which, as a consequence, avoids invalidity of the intermediate subgraphs. Specifically, JT-VAE first decomposes a molecular graph G into a junction tree τ_G to make the graph cycle-free, where each node in the tree represents a substructure of the molecule. Then, both G and τ_G are encoded to latent representations $\mathbf{z}_G$ and $\mathbf{z}_\tau$, respectively, using different encoders. More precisely, $\mathbf{z}_\tau$ encodes the junction tree without capturing the exact mutual connections between substructures, while $\mathbf{z}_G$ encodes the graph to capture the fine-grained connectivities. Afterward, JT-VAE reconstructs the junction tree from its latent representation $\mathbf{z}_\tau$ using a tree-structured decoder, where a tree is generated node-by-node, in a top-down manner. Next, the authors introduce a graph decoder to reproduce the molecular graph based on the underlying predicted junction tree. Since there are potentially many molecules corresponding to the same junction tree, the graph decoder learns how to assemble the subgraphs (nodes in the tree) to reconstruct the molecular graph. Furthermore, to generate molecules with desired properties, the method performs Bayesian optimization in the latent space.

JT-VAE is basically designed to use small substructures as building blocks, which degrades its performance for generating larger molecules such as polymers. To address this issue, HierVAE [37] recently proposes a motif-based hierarchical graph encoder-decoder that employs significantly larger motifs as basic building blocks. To this end, the authors design an encoder that learns hierarchical representation for a given molecular graph G in a fine-to-coarse fashion, from atoms to connected motifs. Then, it obtains the latent vector $\mathbf{z}_G$ by sampling from a Gaussian distribution parametrized using the acquired motif representations. The decoder, on the other hand, autoregressively generates a molecular graph in a coarse-to-fine fashion conditioned on $\mathbf{z}_G$. More specifically, at each generation step, the decoder first predicts the next motif to be attached to the already generated graph. Then, it predicts the attachment points in the new motif, i.e., what atoms belong to the intersection of the new motif and its neighbor motifs. Finally, the decoder decides how the new motif should be attached to the current graph based on its predicted attachment points. The model parameters are learned by minimizing the VAE loss function formulated in Eq. (29). Furthermore, the authors extend the architecture to graph-to-graph translation [62] in order to induce desired properties in the generated molecules.

MHG-VAE [58] guides VAE to always generate valid molecular graphs by proposing *molecular hypergraph grammar* (MHG), a special case of *hyperedge replacement grammar* (HRG) [105] for generating *molecular hypergraphs*, to encode chemical constraints. In particular, the proposed encoder consists of three parts as follows:

$$Enc = Enc_N \circ Enc_G \circ Enc_H, \tag{34}$$

where Enc_H first encodes a molecular graph into a molecular hypergraph, and Enc_G represents the molecular hypergraph as a parse tree by leveraging MHG. Then, Enc_N encodes the previously generated parse tree into the latent continuous space using a seq2seq GVAE [106]. The decoder, on the other hand, acts as an inversion to the encoder and applies production rules, including those with chemical substructure terminals. Finally, using Bayesian optimization, MHG-VAE optimizes the latent continuous space (and its corresponding molecules) towards desired properties.

MoleculeChef [38] proposes to generate molecular graphs using a set of common reactant molecules as building blocks to address the synthesizability issue. In particular, the encoder maps from a multiset of reactants to a distribution over latent space. This is done by using GGNNs [107] to embed each reactant molecule separately, which are further summed to form one embedding for the whole multiset. A feed-forward network is then used to parameterize a Gaussian distribution over the latent space. The decoder, on the other hand, autoregressively maps from the latent space to a multiset of reactants using an RNN, where the latent vector $\mathbf{z}$ initializes its hidden layer, and at each generation step, it outputs one reactant or halts the process. Afterward, a reaction predictor [108] predicts how the previously generated reactants produce a final molecule as illustrated in Figure 8. To learn the model parameters, MoleculeChef minimizes the following WAE [109] objective function:

$$\mathscr{L}_{\theta,\phi}(G) = \mathbb{E}_{G\sim\mathscr{D}}\mathbb{E}_{q_\phi(\mathbf{z}|G)}[-\log p_\theta(G|\mathbf{z})] + \lambda D(\mathbb{E}_{G\sim\mathscr{D}}[q_\phi(\mathbf{z}|G)], p(\mathbf{z})), \quad (35)$$

where D is a divergence measure, namely the maximum mean discrepancy (MMD). Finally, the authors propose to optimize the molecular properties in the continuous latent space in a similar manner to CGVAE [39], which we will discuss in the following subsection.

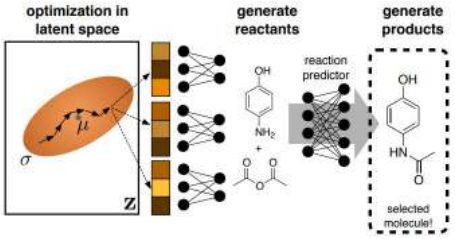

FIGURE 8. An overview of MoleculeChef [38] (reprinted with permission).

C. NODE-BY-NODE GENERATORS

Besides the methods reviewed above, some other autoencoder-based DGGs use graph nodes as building blocks, and their decoders adopt autoregressive generation strategies. CGVAE [39] is one of these methods whose encoder first samples a latent vector $\mathbf{z}_v$ for each node v of an input graph from a normal distribution parametrized using GGNNs [107]. Then, the decoder starts from these vectors and sequentially generates a graph node-by-node with the help of two decision functions, namely, `focus` and `expand`. More precisely, the `focus` function determines the next node to be added into the graph, and the `expand` function iteratively chooses edges to add from the currently focused node until particular stop criteria met, and once the generated subgraph changes, all node representations get updated. Moreover, CGVAE employs a valency masking mechanism as part of `expand` function in the case of molecule generation to guarantee chemical validity. The authors also propose to optimize graph properties by minimizing L_2 distance between some numerical property Q and a differentiable gated regression score $R(G)$ as formulated in Eq. (36), using gradient ascent in the continuous latent space:

$$R(G) = \sum_v \sigma(g_1(\mathbf{z}_v)).g_2(\mathbf{z}_v), \quad (36)$$

where g_1 and g_2 are neural networks.

DEFactor [40] proposes an encoder-decoder based architecture with a recurrent autoregressive decoder for conditional graph generation. In this regard, the encoder first applies a GCN [1] to obtain node embeddings, which are then aggregated by an LSTM to compute the graph-level representation $\mathbf{z}$. Then, the two-step decoder first employs another LSTM in order to autoregressively generate embeddings for each of the graph nodes based on the computed $\mathbf{z}$:

$$h_i = f_{trans}(g_{in}([\mathbf{z}, s_{i-1}]), h_{i-1}), \quad s_i = f_{embed}([h_i, \mathbf{z}]), \quad (37)$$

where f_{trans} is implemented by an LSTM [86], g_{in} and f_{embed} are MLPs, and s_i is the node embedding generated at timestep i. Next, the decoder establishes an edge factorization approach to compute the existence probability of an edge of type k between nodes u and v as follows:

$$p(E_{u,v,k}|s_u, s_v) = \sigma(s_u^\intercal D_k s_v), \quad (38)$$

where D_k is the diagonal matrix of learnable factors for the k-th edge type. Finally, DEFactor makes the generation process conditional by first concatenating the condition vector C with $\mathbf{z}$. It then utilizes a pre-trained discriminator to assess the property C in the graphs generated by the decoder in the training phase.

NeVAE [59] proposes a probabilistic and permutation invariant encoder that is relatively similar to other graph representation learning algorithms, such as GraphSAGE [2] and GCNs [1], except that it uses variational inference to learn the aggregator functions, which is further proved that makes the resulting embeddings well suited for the molecular graph generation task. The authors then introduce a probabilistic decoder that first samples the number of graph nodes from a Poisson distribution. It also samples a latent vector $\mathbf{z}_v$ per node $v \in V$ from $\mathcal{N}(\mathbf{0}, \mathbf{I})$. Then, for each node v, the decoder passes $\mathbf{z}_v$ through a neural network followed by a softmax classifier to determine node features, i.e., the atom type. Next, the total number of graph edges is sampled from a Poisson distribution parametrizes by another neural network conditioned on all latent vectors $\mathbf{Z}$. Thereafter, the decoder samples graph edges one by one from a softmax distribution among all potential edges not generated that far, and similar to CGVAE [39], uses a set of binary masks to guarantee some local structural and functional properties. Finally, it determines the edge type by sampling from another softmax distribution with different binary masks. These masks get updated every time the decoder generates a new edge. The model's objective is similar to that of conventional VAE-based methods plus maximizing the Poisson distribution log-likelihood, which models the number of graph nodes. Moreover, similar

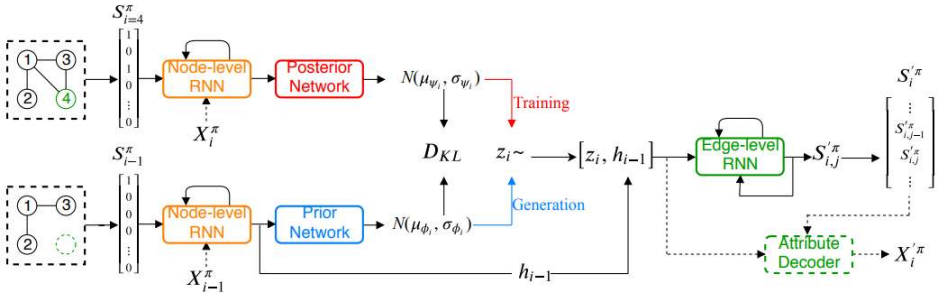

FIGURE 9. The framework of GraphVRNN [41] (reprinted with permission).

to JT-VAE [57], NeVAE utilizes Bayesian optimization over the continuous latent space to discover molecules with desirable properties.

GraphVRNN [41] proposes a VAE-based extension to GraphRNN [25] to learn the joint probability distributions of graph structure as well as the underlying node attributes by rewriting the likelihood function in Eq. (3) as follows:

$$p(S^{\pi}, X^{\pi}) = \prod_{i=1}^{n+1} p(S_i^{\pi}, X_i^{\pi} | S_{<i}^{\pi}, X_{<i}^{\pi}), \tag{39}$$

where $X \in \mathbb{R}^{n \times k}$ is the attribute matrix. Then, the authors adopt an autoregressive variational autoencoder to capture the latent factors over graphs with complicated structural dependencies by optimising the lower bound of the likelihood as follows:

$$\mathscr{L}_{\theta,\phi,\psi}(S^{\pi}, X^{\pi}) = \sum_i \mathbb{E}_{\mathbf{z}_i \sim q_{\psi}(.)}[\log p_{\theta}(S_i^{\pi}, X_i^{\pi} | S_{<i}^{\pi}, X_{<i}^{\pi}, \mathbf{z}_{\leq i})] - \beta D_{KL}(q_{\psi}(\mathbf{z}_i | S_{\leq i}^{\pi}, X_{\leq i}^{\pi}) || p_{\phi}(\mathbf{z}_i | S_{<i}^{\pi}, X_{<i}^{\pi})), \tag{40}$$

where $q_{\psi}(\mathbf{z}_i | S_{\leq i}^{\pi}, X_{\leq i}^{\pi})$ and $p_{\phi}(\mathbf{z}_i | S_{<i}^{\pi}, X_{<i}^{\pi})$ are the proposal and the prior distributions in conditional VAE (CVAE) [110] formulation, respectively, and D_{KL} is the Kullback-Leibler (KL) divergence that is tuned by the β hyperparameter. An overview of the GraphVRNN framework is depicted in Figure 9.

Lim *et al.* [42] utilize a combination of VAE and DeepGMG [33] for generating molecular graphs with desired properties containing an arbitrary input scaffold in their structure. To this purpose, the encoder uses a variant of the interaction network [100], [111] to obtain a representation vector h_G for the input graph, which will be further used to parametrize a normal distribution to sample a latent vector $\mathbf{z}$. The decoder, on the other hand, takes a scaffold S as input and extends it by making sequential decisions of node and edge additions in the same way as DeepGMG [33], except that it also incorporates the latent vector $\mathbf{z}$ in the graph propagation process so that updating node and edge features during the generation is directly affected by $\mathbf{z}$. Moreover, the model makes it possible to conduct the process towards generating molecules with desired properties by concatenating the corresponding condition vector C with $\mathbf{z}$. After the training with the VAE objective finishes, one could give a scaffold S as well as a condition vector C concatenated with a $\mathbf{z}$ sampled from the standard normal distribution to the decoder in order to get a generated molecule with optimized properties.

V. RL-BASED DEEP GRAPH GENERATORS

This section provides a detailed review of generative approaches utilizing reinforcement learning algorithms to induce desired properties in the generated graphs. Table 5 gives a comparison among these methods from multiple aspects.

GCPN [43] proposes a stepwise approach for molecular graph generation, formulating the problem as a Markov Decision Process to train an RL agent in a chemistry-aware environment. Thus, at each generation step t, the method first takes the intermediate graph G_t and the set of scaffolds C as input and computes the state s_t by applying a GCN variant that supports multiple edge types. It then samples an action a_t from the policy π_{θ} based on the obtained node embeddings, which can be either to add a new scaffold subgraph or connect two nodes already in the graph. Next, the action will be further processed by the state transition dynamics, and if it violates chemical rules, it will be rejected so that the state stays unchanged. After that, GCPN utilizes two types of rewards to guide the RL agent, namely, intermediate and final rewards, where the former consists of a stepwise validity reward that encourages the generation process to obey chemical valency rules, and an adversarial reward, which employs the GAN framework [64] to ensure similarity between real molecules and those to be generated. On the other hand, the final reward includes a domain-specific reward for molecular property optimization and a similar adversarial reward. Finally, the authors adopt Proximal Policy Optimization (PPO) [112] to optimize the policy network parameters. An overview of the method is depicted in Figure 10, where each row corresponds to one step in the generation process.

Subsequently, several methods propose extensions to GCPN. For example, Shi *et al.* [44] utilize a combination of general semantic features extracted from the SMILES representations of molecules and their graph representations in order to form more comprehensive states during

TABLE 5. The main characteristics of rl-based deep graph generators.

Method	Generation Strategy	Attention Mechanism	Features	Conditional Generation	Action	Reward
GCPN [43]	Sequential	No	Node/Edge	No	Link prediction	Domain-specific + GAN
Shi et al. [44]	Sequential	Yes	Node/Edge	No	Link prediction	Domain-specific
Karimi et al. [45]	Sequential	Yes	Node/Edge	Yes	Link prediction	Domain-specific + GAN + A reward for drug combinations
DeepGraphMolGen [46]	Sequential	No	Node/Edge	No	Link prediction	Domain-specific + GAN + Property reward (by the property prediction network)
GraphOpt [47]	Sequential	No	Node/Edge	No	Link prediction	Learning a reward function via inverse reinforcement learning
MNCE-RL [48]	Sequential	No	Node/Edge	No	Production rule selection	Domain-specific + A reward regarding the number of generation steps
GEGL [49]	Sequential	No	Node/Edge	No	Generating a molecule	Domain-specific

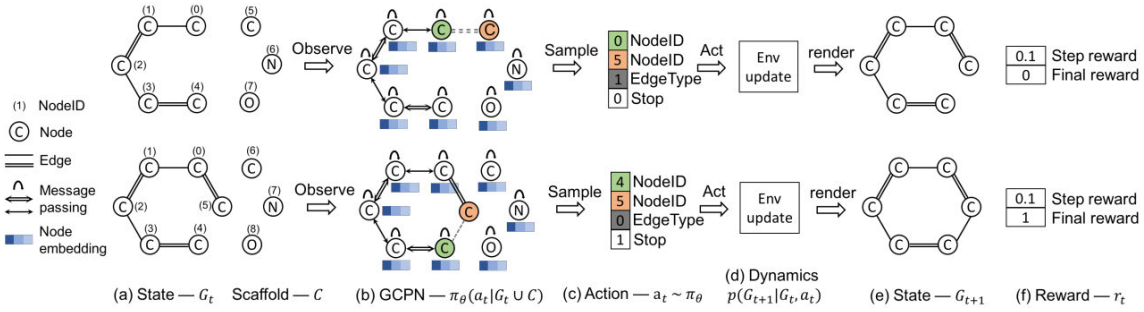

FIGURE 10. An overview of GCPN [43] (reprinted with permission).

the generation process. To this end, the authors propose an architecture consisting of a SMILES encoder and an action generator. First, the encoder obtains context vector z from an input SMILES string, which will be further processed by two attention mechanisms, namely action-attention and graph-attention, to get the enhanced context vector $\tilde{z}$. Then, the model concatenates the current graph state s_t with $\tilde{z}$ to pass a heterogeneous state into the action generator that has the same generation mechanism as GCPN, except that it does not involve adversarial rewards. Furthermore, the model is trained in two stages. The supervised learning stage learns an initialization for the model parameters to alleviate the instability of an RL agent training by minimizing the following objective function:

$$J = -\frac{1}{M}\sum_{m=1}^{M}\log\frac{1}{N}\sum_{n=1}^{N}\sum_{t}\log P(a_t|\tilde{z}, s_t) + D_{KL}(P_z||P_0), \tag{41}$$

where M is the number of molecules in the training dataset, N denodes the number of sampled trajectories for generating each molecule, and P_z and P_0 are the distribution of the learned context vector and a prior distribution, respectively. Afterwards, the reinforcement learning stage further optimizes the process towards generating molecular graphs with desired properties. Figure 11 provides an overview of this framework.

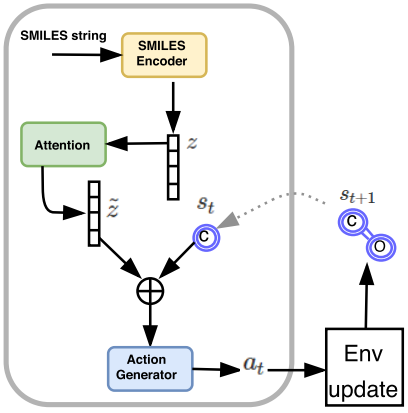

FIGURE 11. An overview of the framework proposed by Shi *et al.* [44]. In the supervised learning phase, only the part in the gray box is trained. On the other hand, the whole architecture is involved in the reinforcement learning stage.

Karimi *et al.* [45] propose another extension to GCPN for drug-combination design, which is a key part of combination therapy. To this end, the authors first develop Hierarchical Variational Graph Auto-Encoders (HVGAE) to embed prior knowledge such as gene-gene, gene-disease, and

disease-disease networks to acquire more accurate disease representations. Then, they formulate the problem as generating a set of graphs $\mathcal{G} = \{G^{(k)}\}_{k=1}^{K}$ conditioned on the learned disease representations by employing a similar generation strategy as GCPN, with the difference that in addition to chemical validity and adversarial rewards, they design a reward to encourage generating disease-specific drug combinations.

More recently, DeepGraphMolGen [46] combines GCPN with a molecular property prediction network implemented as a GCN followed by a feedforward layer, which provides GCPN with an extra chemical reward to tilt the process towards generating molecules with additional property, i.e., binding potency to dopamine transporters.

Besides GCPN and its subsequent approaches, there exist other RL-based methods that generate graph structures by taking different strategies. GraphOpt [47] models graph formation via a Markov Decision Process, aiming to learn both a graph construction procedure Π and a usually unknown latent objective function $\mathscr{F} : G \to \mathbb{R}$ that reflects the underlying graph formation mechanism. Therefore, inspired by [113], the authors formulate the following objective:

$$\begin{aligned} \Pi^* &= \underset{\Pi}{\operatorname{argmin}} \max_{\mathscr{F}} [\mathscr{F}(G) - \mathscr{F}(\Pi(V))], \\ \mathscr{F}_{opt} &= \underset{\mathscr{F}}{\operatorname{argmax}} [\mathscr{F}(G) - \mathscr{F}(\Pi^*(V))], \end{aligned} \tag{42}$$

where $\mathscr{F}_{opt}$ assigns the highest score to the observed graphs compared to all other ones, and optimization over $\mathscr{F}$ is, in fact, a search for the reward function via inverse reinforcement learning (IRL) [114]. In other words, GraphOpt learns a reward function, which is in contrast to most of the RL frameworks that utilize an existing one. The optimal construction procedure, on the other hand, tries to construct a graph $G' = \Pi^*(V)$ in a sequential link formation process given node-set V, which is expected to be the most similar graph to the observed one using $\mathscr{F}$ as the similarity measure. To this end, the authors propose a continuous latent action space to infer a link formation action a_t at each time step t by first sampling two vectors $a^{(1)}$ and $a^{(2)}$ from a normal distribution parametrized based on the current graph state s_t, which is computed by a GNN [98]. They then choose two graph nodes with the most similar embeddings to the obtained vectors to construct an edge.

MNCE-RL [48] proposes a graph convolutional policy network with a novel GCN architecture for generating molecules with optimized properties, which, similar to MHG-VAE [58], utilizes grammar rules to guarantee the validity of molecules. To this end, the authors first extend the NCE graph grammar [115] to make it applicable for generating molecules. They then infer the production rules of the grammar from a set of input molecules. Next, the RL-based generation process starts whose action space consists of the set of legal production rules, and at each step, the policy samples a rule based on the node features obtained by applying the proposed GCN on the intermediate graph. A domain-specific reward guides the process towards generating desirable molecules. Moreover, MNCE-RL assigns a negative reward when the number of steps exceeds a threshold to avoid prolonging the generating process.

GEGL [49] proposes to incline a deep neural network called neural apprentice policy towards generating molecules with desired properties. In this respect, the apprentice policy first generates a set of molecules by a SMILES-based LSTM and stores them into a fixed-size max-reward priority queue $\mathscr{Q}$. Then, a genetic expert policy utilizes the content of $\mathscr{Q}$ as seed molecules and applies two genetic operators, namely, the graph-based mutation and crossover [116], to them and stores the generated molecules in another priority queue denoted by $\mathscr{Q}_{ex}$. After generating each sample, both $\mathscr{Q}$ and $\mathscr{Q}_{ex}$ are updated so that they always contain molecules with the highest rewards. Next, the apprentice policy updates its model's parameters by learning to imitate the molecules stored in $\mathscr{Q} \cup \mathscr{Q}_{ex}$, and the whole procedure repeats iteratively. This way, the expert policy guides the apprentice policy to generate molecules with preferred properties.

VI. ADVERSARIAL DEEP GRAPH GENERATORS

This section reviews methods employing generative adversarial networks (GANs) [64] to generate either molecular or non-molecular graph structures. To conduct a more accurate study, we divide the existing approaches into multiple subsections. Moreover, Table 6 provides a multifaceted comparison of them.

A. RANDOM WALK-BASED METHODS

A series of works focus on generating random walks instead of the entire graph, as graph random walks are invariant under node reordering. In this respect, NetGAN [65] introduces the first implicit generative model for graphs that learns the distribution of biased random walks over a single graph using the WGAN framework [117]. In particular, NetGAN first samples a collection of random walks using the biased second-order strategy [118] to prepare the model's training data. Then, the generator learns to sequentially generate random walks node-by-node using the LSTM [86] architecture, which is initialized by a latent vector $\mathbf{z}$ sampled from a standard normal distribution. Meanwhile, the discriminator decides whether a random walk is real or not after processing its entire node sequence by another LSTM. After training finishes, the authors construct the adjacency matrix of a new graph using multiple generated random walks. Further to this, a number of generative approaches have been proposed inspired by the idea of NetGAN or extending it. For example, STGGAN [119] adopts a similar generating scheme for spatial-temporal graphs.

MMGAN [66] generalizes NetGAN to capture higher-order connectivity patterns by introducing multiple types of random walks, each biased towards different motif structures. To simplify the process, MMGAN focuses only on 3-node motifs and proposes an architecture consisting of three GANs, namely, NetGAN that considers pairwise relationships, and two other motif-based GANs. The random walks generated

TABLE 6. The main characteristics of adversarial deep graph generators.

Method	Input	Generation Strategy	Attention Mechanism	Features	Conditional Generation
NetGAN [65]	one single graph	Sequential	No	-	No
MMGAN [66]	one single graph	Sequential	No	-	No
SHADOWCAST [67]	one single graph	Sequential	No	Node	Yes
MolGAN [63]	dataset of graphs	All at Once	No	Node/Edge	No
CONDGEN [61]	dataset of graphs	All at Once	No	-	Yes
TSGG-GAN [68]	dataset of graphs	All at Once	No	Node	Yes
VJTNN + GAN [62]	dataset of graphs	Sequential	Yes	Node/Edge	No
Mol-CycleGAN [69]	dataset of graphs	Sequential	No	Node/Edge	No
Misc-GAN [70]	one single graph	Not mentioned	No	-	No

by each of the three GANs are then combined to construct the output graph.

SHADOWCAST [67] proposes another extension to NetGAN in order to make the generation process controllable, which can be considered as a step towards generating graphs with more explainable properties. To this end, the authors first define a graph called *shadow* with the same structure as the original one but with different node labels so that these node-level properties can control the generation process. Then, they expand the architecture of NetGAN by adding a sequence-to-sequence model called shadow caster, which is implemented by an LSTM [86]. Specifically, the shadow caster takes in sampled walks from the shadow network and generates synthetic shadow walks of preferred distribution to control the generation process. Next, these model-generated shadow walks are fed into both generator and discriminator as conditions, which are finally trained using the conditional GAN [120] framework.

B. GRAPH-BASED METHODS

Unlike random walk-based approaches, most of the existing adversarial graph generators deal with the entire graph. Here, we study these methods in two categories.

1) GENERAL GRAPH-BASED ADVERSARIAL DGGs

MolGAN [63] proposes the first implicit generative model for small molecular graphs. In this respect, its generator first takes a latent vector $\mathbf{z}$ sampled from $\mathcal{N}(0, I)$. Then, it outputs a probabilistic graph all at once using an MLP in a way similar to GraphVAE [51], which, as a consequence, limits the model to generate graphs of a predefined maximum size. However, in contrast to GraphVAE, MolGAN does not need to perform an expensive graph matching algorithm, as it makes the model likelihood-free using the GAN framework. Next, a permutation-invariant discriminator tries to distinguish between generated graphs and real ones using a combination of the Relational-GCN [121] and an MLP. The authors train the discriminator using the WGAN [117] objective, while they combine a reinforcement learning objective with that of the WGAN to train the generator, aiming at inclining the process towards generating molecules with desired chemical properties. More precisely, the authors employ a deterministic policy gradient algorithm, namely, DDPG [122], to maximize the reward, which is approximated by a reward network with the same architecture as the discriminator. The overall architecture of MolGAN is shown in Figure 12.

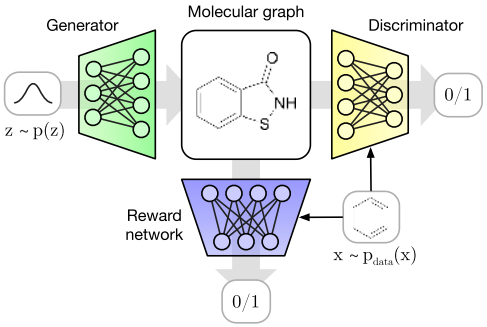

FIGURE 12. An overview of MolGAN [63] (reprinted with permission).

LGGAN [123] adopts a similar generator to that of MolGAN. However, its discriminator uses JK-Net [124] to compute graph embeddings and outputs both the graph label and the probability of the graph being real. Moreover, to incorporate the class information, the authors utilize the AC-GAN [125] framework.

CONDGEN [61] proposes a model of graph variational generative adversarial nets for conditional structure generation. It addresses both the challenges of permutation-invariance and context-structure conditioning indicated by the information of attributes or labels in the networks. The method first applies the trick of latent space conjugation to the base VGAE [50] model in order to convert its node-level encoding into a permutation-invariant graph-level one that allows learning from a dataset of graphs with variable sizes, which is a notable improvement over the VGAE. More precisely, μ and σ in Eq. (27) are replaced by the following parameters:

$$q(\mathbf{z}_i|X, A) = \mathcal{N}(\bar{\mathbf{z}}|\bar{\mu}, \text{diag}(\bar{\sigma}^2)), \tag{43}$$

where $\bar{\mu} = \frac{1}{n}\sum_{i=1}^{n} g_\mu(X, A)_i$ and $\bar{\sigma}^2 = \frac{1}{n^2}\sum_{i=1}^{n} g_\sigma(X, A)_i^2$. However, the process is still not completely permutation-invariant because the reconstruction loss of the VGAE is computed between the generated adjacency matrix A' and the original matrix A, which may be under different node

permutations. Therefore, the authors propose a GCN-based discriminator to enforce the structural similarity between the generated and the true adjacency matrices and learn its parameters by optimizing the GAN objective. Thus, the encodings $\text{GCN}_D(A)$ and $\text{GCN}_D(A')$ computed by the discriminator, become permutation-invariant and the reconstruction loss can be computed as $||\text{GCN}_D(A) - \text{GCN}_D(A')||_2^2$. Moreover, according to [120], the authors use the concatenation of condition vector C and latent variable $\mathbf{Z}$ to enable conditional structure generation. Then, motivated by CycleGAN [126], they further enforce mapping consistency between the graph context and the structure spaces by sharing the parameters in the two GCN networks, namely, the GCNs in the graph encoder and the discriminator.

More recently, TSGG-GAN [68] adopts a time series conditioned generative model that aims to generate a graph given an input multivariate time series, where each time series acts as context information associated with one of the graph nodes. This is particularly the case when it is straightforward to obtain node-level information, while the underlying network is totally unknown. To this end, using SRU [127] to extract the information of the time series followed by an MLP, the generator generates the entire graph all at once. At the same time, the discriminator takes a pair of a multivariate time series and a graph as inputs, which are then processed using SRU and GCN, respectively. Thereafter, the discriminator utilizes Neural Tensor Networks (NTN) [128] to measure the similarity between the time series and the graph and decides whether the graph is real or not.

2) GRAPH-TO-GRAPH TRANSLATORS

In addition to the aforementioned methods, there exist other approaches trying to generate a new graph based on an initial one. VJTNN [62] proposes a graph-to-graph translation model that learns a mapping from a source molecular graph X to a target graph Y with enhanced chemical properties by utilizing a similar encoder-decoder architecture as JT-VAE [57] whose tree decoding process is further enriched by adding an attention mechanism. More specifically, VJTNN augments the basic encoder-decoder model with latent code $\mathbf{z}$ derived based on the embeddings of both source and target graphs and minimizes the conditional VAE loss function to learn the mapping $F : (X, \mathbf{z}) \rightarrow Y$. The authors then propose an adversarial variation called VJTNN + GAN to force generated graphs to follow the distribution of the target ones, which is trained using the WGAN framework [117].

Mol-CycleGAN [69] establishes structural similarity between the source and target molecular graphs by adopting a CycleGAN-based [126] approach. To this end, the method first computes the latent space embeddings for X and Y using JT-VAE [57] and then learns the transformation $F : X \rightarrow Y$ (and its reverse, i.e., $G : Y \rightarrow X$) in that space. Mol-CycleGAN also introduces the discriminator D_X (and D_Y) to decide whether a sample is from the distribution of X (or Y) or it is generated by G (or F). The model parameters are trained by optimizing the following loss function:

$$\begin{aligned}&\mathscr{L}(F, G, D_X, D_Y)\\ &\quad = \mathscr{L}_{GAN}(F, D_Y, X, Y) + \mathscr{L}_{GAN}(G, D_X, Y, X)\\ &\qquad + \lambda_1 \mathscr{L}_{cyc}(F, G) + \lambda_2 \mathscr{L}_{identity}(F, G),\end{aligned} \tag{44}$$

where the authors utilize the adversarial loss of LS-GAN [129] and the similar $\mathscr{L}_{cyc}(F, G)$ and $\mathscr{L}_{identity}(F, G)$ as CycleGAN, where the former reduces the space of mapping functions and acts as a regularizer, while the latter makes the generated molecule not to be structurally far away from the original one. After the training finishes, Mol-CycleGAN takes a molecule X as input and calculates its embedding by applying the encoder of the JT-VAE. Then, $F(X)$ computes an embedding corresponding to a molecule with desired properties that is also structurally similar to X. Finally, the model generates the optimized molecular graph Y using the JT-VAE's decoder.

Misc-GAN [70] proposes another translation model inspired by CycleGAN [126] to learn a mapping function F from a source graph G_s to its corresponding target graph G_t while preserving the hierarchical graph structures (i.e., the community structures) in the target graph in different levels of granularity. The model training consists of three stages: First, it constructs coarser graphs in L granularity levels based on an input target graph G_t. Then, at each level l, it trains an independent CycleGAN-based generative model from G_s to $G_t^{(l)}$. Finally, all generated graphs $\tilde{G}_t^{(l)}$ are aggregated together to form the reconstructed target graph $\tilde{G}_t$. The framework is trained by minimizing the following loss function:

$$\mathscr{L} = \mathscr{L}_{ms} + \mathscr{L}_F + \mathscr{L}_G + \mathscr{L}_{cyc}, \tag{45}$$

where $\mathscr{L}_{ms}$ is the multi-scale reconstruction loss between the target graph G_t and the generated graph $\tilde{G}_t$, $\mathscr{L}_F$ is the forward adversarial loss for learning a mapping from the source to the target graph, $\mathscr{L}_G$ is the backward adversarial loss to learn the reverse mapping, and $\mathscr{L}_{cyc}$ is the cycle consistency loss [126].

VII. FLOW-BASED DEEP GRAPH GENERATORS

In addition to the methods we have discussed so far, a line of research has recently emerged, which employs flow-based approaches in the field of graph generation. For example, GNF [60] develops a generative model of graphs by combining normalizing flows with a graph auto-encoder. More specifically, the authors first train a permutation invariant graph auto-encoder that encodes an input graph to a set of node features $X \in \mathbb{R}^{n \times k}$ using a standard GNN. Then, a simple decoder outputs a probabilistic adjacency matrix $\hat{A}$, in which edge probability between two arbitrary nodes i and j with embedding vectors x_i and x_j is computed as follows:

$$\hat{A}_{ij} = \frac{1}{1 + \exp(C(||x_i - x_j||_2^2 - 1))}, \tag{46}$$

where C is a temperature hyperparameter. After the auto-encoder training completes, the encoder is employed to

compute node features X to be used as training input for the GNF. Then, the GNF, which is based on non-volume preserving flows [130], learns a mapping from the complicated graph distribution into a latent distribution that is well modelled as a Gaussian. At inference time, GNF generates node features by first sampling $Z \sim \mathcal{N}(0, I)$ from the latent space followed by applying the inverse mapping, $X = f^{-1}(Z)$ which is then fed into the decoder to get the predicted adjacency matrix as illustrated in Figure 13. GraphNVP [71] takes a similar approach, but rather than pretraining an auto-encoder to get continuous node features, the authors propose to perform Dequantization [130], [131] by adding uniform noise to the discrete adjacency tensor as well as the node label matrix. More precisely, GraphNVP proposes a two-step generation scheme by learning two latent representations for each graph, one for the adjacency tensor and the other for node labels.

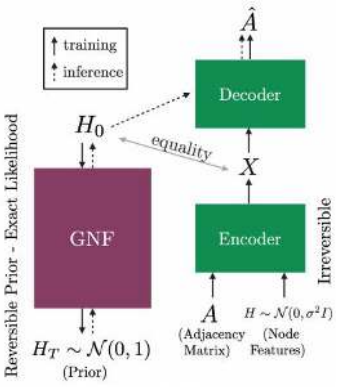

FIGURE 13. The framework of GNF [60] for the graph generation (reprinted with permission).

In addition to the flow-based methods we have studied so far that generate the whole graph in one step, there are other approaches adopting autoregressive generation strategies. For example, GraphAF [35] proposes to generate molecular graphs by combining the advantages of both autoregressive and flow-based models. The method first converts a molecular graph structure $G = (A, C)$, where both node type matrix C and the adjacency matrix A are discrete, into continuous data $G' = (A', C')$ using Dequantization technique [130], [131] in order to make the data usable for a flow-based model. Then, conditional distributions for the i-th generation step are defined according to Autoregressive Flows (AF) [132] as follows:

$$\begin{aligned} p(C'_i|G_i) &= \mathcal{N}(\mu_i^C, (\alpha_i^C)^2) \\ p(A'_{ij}|G_i, C_i, A_{i,1:j-1}) &= \mathcal{N}(\mu_{ij}^A, (\alpha_{ij}^A)^2), \end{aligned} \tag{47}$$

where G_i is the current sub-graph, μ_i^C, α_i^C, and μ_{ij}^A, α_{ij}^A are the means and standard deviations of Gaussian distributions, which are computed by different neural networks based on node embeddings of the sub-graph generated so far. To calculate the exact likelihood, an invertible mapping from the molecule structures $G' = (A', C')$ to latent Gaussian space $\mathbf{z}$ is defined as:

$$z_i = (C'_i - \mu_i^C) \odot \frac{1}{\alpha_i^C}, \quad z_{ij} = (A'_{ij} - \mu_{ij}^A) \odot \frac{1}{\alpha_{ij}^A}, \tag{48}$$

where $\frac{1}{\alpha_i^C}$ and $\frac{1}{\alpha_{ij}^A}$ denote element-wise reciprocals of α_i^C and α_{ij}^A, respectively and $\odot$ is the element-wise multiplication. At inference time, GraphAF just samples random variables z_i and z_{ij} from the latent Gaussian space and converts them to the molecule structures as in Eq. (49) to generate new graphs in an autoregressive manner:

$$C'_i = z_i \odot \alpha_i^C + \mu_i^C \;\; A'_{ij} = z_{ij} \odot \alpha_{ij}^A + \mu_{ij}^A. \tag{49}$$

GraphAF further proposes a valency-based rejection sampling similar to MolecularRNN [26] to guarantee the validity of generated molecules. The authors also propose to fine-tune the generation process with reinforcement learning to generate molecules with optimized properties.

More recently, GrAD [36] proposes another autoregressive flow-based approach for graph generation, which can also be considered as a variant of GRAN [30]. In particular, its training consists of two stages. Firstly, for generating each new block of B nodes in the t-th step, the model samples latent codes $H_{b_t} \in \mathbb{R}^{B \times k}$ to initialize the corresponding node representations. Then, different from GRAN's formulation in Eq. (18), the node features get updated using graph attention layers almost according to [89], except that self-attention weights are calculated only based on each node's neighborhood to inject structural information of the currently generated graph into the updating process. After node representations are obtained, the model follows a similar generation strategy as GRAN and jointly optimizes the generator's parameters and the distribution of latent codes. In the second training stage, GrAD trains a flow-based reversible model to map samples from the optimized distribution of latent codes to a simple Gaussian distribution. Therefore, at the inference time, one can first sample a batch of B vectors $Z \in \mathbb{R}^{B \times k}$ from a Gaussian base distribution and apply the inverse mapping $H_{b_t} = f^{-1}(Z)$ to obtain initial node representations and then go through the generation process.

Table 7 summarizes the main characteristics of the flow-based DGGs.

VIII. DEEP GRAPH GENERATOR CATEGORIES: SUMMARY AND COMPARISON

So far, we have divided the existing DGGs into five overlapping categories and reviewed the methods in each category in detail. In this section, we summarize the characteristics of the categories mentioned in Sections III to VII and compare them against each other.

We first discuss autoregressive DGGs. The main characteristic of these methods is that they adopt a step-by-step strategy for graph generation such that at each stage, decisions are made based on the results obtained from the previous steps. In addition, based on whether the whole generation history influences the current decision or the decision is

TABLE 7. The main characteristics of flow-based deep graph generators.

Method	Category	Generation Strategy	Attention Mechanism	Features	Conditional Generation
GNF [60]	Autoencoder-based	All at Once	Yes	Node/Edge	No
GraphNVP [71]	-	All at Once	No	Node/Edge	No
GraphAF [35]	Autoregressive	Node-by-node	No	Node/Edge	No
GrAD [36]	Autoregressive	Block of nodes	Yes	-	No

made based only on the graph generated so far, we further divide autoregressive methods into two sub-categories: recurrent and non-recurrent. The advantage of the algorithms in this category over other categories is their stepwise nature of generating process, which potentially makes them more scalable. Therefore, the most scalable DGGs [25], [29], [30], [34], generating the largest graphs with several thousands of nodes belong to this category, which are trained on datasets of graphs instead of a single graph. Furthermore, there are approximately equal numbers of autoregressive methods for molecular and non-molecular applications, that will be discussed in the following section.

Autoencoder-based DGGs generate graphs from a continuous latent space. While most of them utilize VAEs, only a limited number of them use simple autoencoders. These methods first encode a graph into a latent space and then generate the desired graph from this space. Most of the autoencoder-based graph generators proposed so far are not scalable enough to be applied on datasets of large graphs. They typically can generate graphs with less than 100 nodes, which are more of the molecular type. However, there exist a few methods [50], [54] that are trained on one single relatively large graph for the task of link prediction. Furthermore, this type of DGGs can optimize different objectives in the continuous latent space and generate graphs with desired properties by decoding the optimized latent vectors. This is usually easier than performing the optimization in the graph space, which is another reason why these approaches are mostly used for generating molecular graphs with certain chemical properties. Moreover, the graph generators that have taken steps towards model interpretability and disentanglement, belong to this category.

The third category, which has fewer methods than the previous two ones, is dedicated to the RL-based DGGs. These methods aim to learn a graph generation policy by maximizing a reward function, which unlike other approaches, allows performing the optimization directly in the graph space, based on a criterion that is not necessarily differentiable. Hence, almost all the RL-based DGGs are designed to generate molecular graphs with specific properties induced by the reward function which have commonly less than 100 nodes. Furthermore, as most of the current RL-based graph generators adopt a step-by-step generation strategy, they also belong to the autoregressive category. However, they do not employ just a maximum likelihood-based objective function as opposed to the methods reviewed in Section III.

Adversarial DGGs employ generative adversarial networks [64] for generating graph structures. This sidesteps the need for likelihood-based optimization that is a challenge for graph generation tasks due to the factorial number of node orderings in terms of the number of graph nodes, which makes optimizing the exact likelihood function intractable. Some of the existing adversarial DGGs focus on generating random walks and train their generative model by using only one single and large graph (commonly with thousands of nodes), while others generate the entire graph, which are mostly trained on datasets of small graphs (with fewer than 200 nodes). In addition, these methods have been used almost equally in molecular and non-molecular applications.

Recently, a small number of flow-based DGGs have been proposed. They make the exact likelihood optimization possible by defining invertible transformations from the complicated data distribution to a simple distribution. However, defining an invertible mapping function adds complexity to the design of these models. In addition, flow-based graph generators are applied in both molecular and non-molecular applications with more concentration on the former and therefore they have been utilized most often on rather small graphs with fewer than 100 nodes.

IX. APPLICATIONS

Deep graph generation approaches have a wide range of applications, from discovering new molecular structures and building knowledge graphs to modeling physical, social, and biological networks. Here we review some of the most explored applications and suggest potential future directions.

A. MOLECULAR GRAPH GENERATION

The molecule generation approaches aim to downsize the high-dimensional chemical space to expedite drug design and material discovery. Since molecules can be considered as graphs, where the atoms form the graph's node-set and the chemical bonds determine how those nodes connect, the most widely explored application of modern deep graph generative methods is generating molecular structures, a problem that has been previously mostly formulated as producing SMILES strings [106], [133]–[136]. Using graph generators instead of SMILES based models results in generating more valid intermediate substructures compared to mostly meaningless partially generated substrings. It also allows better capturing the similarities between molecules as molecules with similar structures can have totally different SMILES encodings.

Deep molecular graph generation approaches proposed so far utilize various frameworks, from VAEs [38], [39], [51]–[53], [57]–[59] and GANs [37], [63], [69] to RL-based [43], [44], [48], autoregressive [24], [26], [31], [33], [35], [42] and flow-based frameworks [35], [60], [71]. They also adopt different generation strategies at varying granularity levels. For example, some of them generate the whole graph all at once [51], [63], while others add one atom at a time [24], [26], [33] or use valid chemical substructures as their building blocks [37], [42], [57].

Moreover, in molecular graph generation, two challenges must be taken into account. First, the generated molecules must satisfy the explicitly specified validity constraints, i.e., an atom's chemical bonds should not exceed its valence. The graph generator models proposed so far address the issue by employing different mechanisms, including introducing structural penalties during the training [26], [43], adopting valency-based rejection sampling at the inference time [26], [35], utilizing a grammar-based approach [58], adding regularization terms to the objective function [53], using valid chemical substructures as building blocks [37], [38], [42], [57], and employing valency masking mechanism [39], [59]. The second challenge to be considered is that new molecular structures should obey some desired properties. This problem has also been addressed by taking various strategies including minimizing a distance [38], [39] or utilizing Bayesian optimization [57]–[59] in some continuous latent space, maximizing a domain-specific reward in RL-based approaches [43], [44], [63], or performing the generation given an input molecule with desired properties and try to preserve those properties in the target molecule [37], [69].

B. NON-MOLECULAR GRAPH GENERATION

Although the most remarkable application of modern graph generation approaches explored so far is generating molecular structures, several other non-application-specific approaches have been proposed [25], [28], [30], [36], [41], [55], [61] working on more general datasets. While these methods' ultimate goal is to be used on real-world applications such as generating social network graphs, most of them suffer from scalability issues. Thus, they are currently being applied to synthetic or relatively small real datasets. Despite the steps taken towards making these models more scalable [25], [30], [34], it should be specifically considered as future work.

C. FUTURE APPLICATIONS

Beyond the discussed applications, some other problems can potentially be solved from a graph generation perspective. This is particularly the case when the output space includes graphs, and so the generated output, on the one hand, must depend on the input and, on the other hand, must obey the distribution of the output space graphs. Below, two practical examples of these problems are mentioned.

1) LANGUAGE-BASED GRAPH GENERATION

In natural language processing, several approaches have been proposed to extract rich graph-structured information from textual data. They include methods aiming to extract AMRs (Abstract Meaning Representations) [137]–[139], semantic graphs [140], semantic dependency graphs [141]–[143], and even those that transform one graph into another based on some input sentences [144]. However, most existing methods often propose some domain-specific procedures to produce these graph-structured knowledge representations, which, as a result, are not capable enough to consider various effective factors. Therefore, as a future orientation, the community can solve such problems with a conditional generative approach, making it possible to generate graphs from their corresponding distribution given the specified textual input.

2) SCENE GRAPH GENERATION

Scene graphs are structured representations of images providing higher-level knowledge for scene understanding, where the objects in each image form the node-set of the corresponding scene graph, and the relationships between objects determine how the nodes connect. As this class of graphs has a wide range of applications, including image captioning, visual question answering, image retrieval, and image generation [145], scene graph generation becomes a line of research in recent years.

Most scene graph generation methods first detect objects from an input image using object detection models like Faster R-CNN [146] to form the set of graph nodes. Meanwhile, the relationships can be extracted either jointly with the objects [147] or after all the objects are detected [148]–[151]. Among these approaches, some generate scene graphs solely based on input images [148], [149], [152], while others benefit from additional text input [150], [153], or some self-generated extra information [145], [151]. Moreover, as the number of objects increases, some models [145], [154], [155] propose to first generate multiple subgraphs and then aggregate them to construct the complete scene graph to address the scalability issue.

Here, we have briefly reviewed and categorized some of the scene graph generation methods. However, they are more of a relational information extractor from images rather than graph generators, which estimate the underlying data distribution. Therefore, it can be explored in the future.

X. IMPLEMENTATIONS

In this section, we discuss the implementation details by categorizing and summarizing commonly used datasets and evaluation metrics. We also collect the available source codes in Appendix A.

A. DATASETS

There are many datasets for learning on graphs that have been investigated by previous studies, including [73] and [156]. However, none of these studies has thoroughly and

TABLE 8. Summary of the commonly used datasets.

Category	Dataset Name	# Graphs	Range of # Nodes	Range of # Edges	# Node Labels	# Edge Labels	Citation
Chemical & Bioinformatics	ZINC	249456	[6, 38]	[5, 45]	9	4	[26], [29], [31], [35], [39], [40], [43], [44], [46], [48], [49], [51], [53], [57]–[59], [62], [69], [71]
	QM9	133885	[1, 9]	[0, 13]	4	4	[35], [39], [51]–[53], [59], [60], [63], [71]
	Polymer	90823	[5, 122]	[5, 145]	7	4	[37]
	USPTO	479035	[17, 258]	[17, 288]	72	4	[38]
	ChEMBL	1913742	[1, 780]	[0, 843]	33	4	[24], [26], [33], [69]
	Protein	1113	[4, 620]	[5, 1049]	3	✗	[25], [28], [30], [31], [34], [36], [55]
	NCI-H23	24k	[6, 50]	[6, 57]	11	3	[29]
	Yeast	47k	[5, 50]	[5, 57]	11	3	[29]
	MOLT-4	33k	[6, 111]	[6, 116]	11	3	[29]
	MCF-7	23k	[6, 111]	[6, 116]	11	3	[29]
	Enzymes	600	[2, 125]	[1, 149]	3	✗	[28], [29]
Social	Cora	1	2708	5278	7	✗	[29], [47], [50], [54], [60], [65]–[67]
	Citeseer	1	3327	4614	6	✗	[29], [47], [50], [54], [65], [66]
	Pubmed	1	19717	44325	3	✗	[47], [50], [54], [60], [65]
	DBLP (V13)	1	5354309	48227950	✗	✗	[61], [65]
Synthetic	Barabasi-Albert	-	-	-	-	-	[25], [31], [33], [47], [54], [56], [68]
	Erdos-Renyi	-	-	-	-	-	[47], [54]–[56]
	Watts-Strogatz	-	-	-	-	-	[55]
	Community	-	-	-	-	-	[25], [28], [31], [35], [36], [41], [60]
	Grid	-	-	-	-	-	[25], [30], [31], [34], [36]
	Lobster	-	-	-	-	-	[31], [34], [36]
	Cycles	-	-	-	-	-	[33], [36]
	Ego	-	-	-	-	-	[25], [28], [31], [32], [35], [36], [41], [54], [56], [60]

exclusively collected and categorized the datasets used in graph generation approaches. Here, we have summarized the most prominent ones in three general categories according to the main graph generation applications, as shown in Table 8. Downloadable links for real-world datasets are accessible by clicking on their names. Synthetic datasets are also provided with links to their generating codes. Furthermore, to give the readers a more detailed view of the data used by DGGs, we provide statistical information of the real-world datasets. This information is based on the latest or the most popular versions of these datasets, which can be accessed via the provided links.

Besides the datasets listed in Table 8, there are two benchmarks related to the molecular graph generation tasks. Below, we provide a brief description of them:

- **GuacaMol** [157] is an evaluation framework (i.e. an open-source Python package) for the assessment of models for de novo molecular design using a subset of the ChEMBL database. MNCE-RL [48] and GEGL [49] utilize GuacaMol in their experiments.
- **MOSES** [158] is another benchmarking platform for standardizing training and comparison of molecular generative models that, similar to GuacaMol, provides an open-source Python package. It uses a subset of the ZINC database for training and testing. MolecularRNN [26] and GraphAF [35] use MOSES for their model evaluation.

As mentioned earlier, we categorize the datasets utilized to evaluate deep graph generators into three classes. Here, we briefly review the characteristics of each class:

- **Chemical & Bioinformatics:** The datasets of this category consist of a large number of graphs (mostly more than tens of thousands and even up to millions of graphs); however, these graphs are relatively small in size, with usually less than 150 nodes. Almost all of these datasets have node and edge labels.
- **Social:** This category contains datasets, each consisting of only one single but large graph (containing from thousands to millions of nodes). These graphs are all citation networks whose nodes correspond to publications, and edges represent the citation relationship between papers. Since these graphs are large, and most current graph generators suffer from scalability problems, several techniques have been employed to deal with this issue. For example, some DGGs focus on generating random walks instead of the entire graph. Then, they construct the final graph by aggregating multiple generated random walks [65]–[67]. Some others sample several subgraphs from the main graph to form their training set [29], [61]. Finally, the remaining approaches base their evaluation more on tasks such as link prediction and node classification [47], [50], [54], [60].
- **Synthetic:** Particular patterns are employed to produce each of the synthetic datasets. Therefore, they are primarily used to assess the ability of generative models

to learn these patterns. The statistics of utilized synthetic datasets can vary significantly from one DGG to another since they can be produced under arbitrary settings. Therefore, summarizing their statistical information is meaningless, and we leave the corresponding fields blank in Table 8

Furthermore, to better analyze data categories mentioned in Table 8, Figure 14 shows the tendency of DGGs belonging to each of the five categories towards using different types of datasets. It reveals that chemical and bioinformatics datasets are the most popular among current DGGs and confirms our discussion in Section IX that indicates generating molecular structures is the most explored application of modern graph generation approaches. More precisely, autoregressive, autoencoder-based, and RL-based graph generators have performed most of their experiments on these datasets. For autoregressive approaches, this is mainly due to the step-by-step nature of the generation process, which allows making wiser and more controlled decisions to produce molecules with desired chemical properties that also satisfy chemical constraints. Additionally, autoencoder-based DGGs are highly appropriate for utilizing this type of dataset since optimizing various objectives in the continuous latent space allows them to generate graphs with desired properties. This is also true for RL-based graph generators whose reward functions are designed to induce specific properties in the generated graphs (for a more detailed discussion please refer to Section VIII).

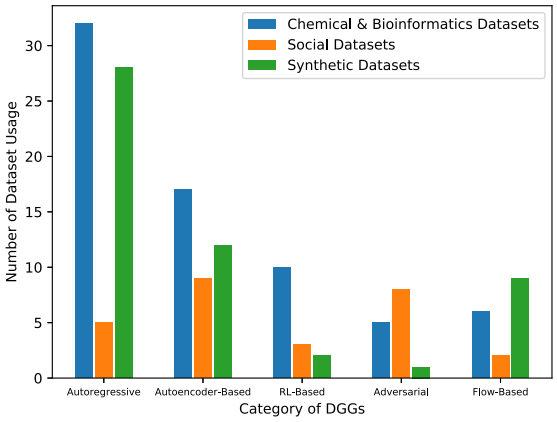

FIGURE 14. The number of dataset usage by DGGs belonging to each of the five categories for each type of dataset (based on the methods referenced in Table 1 and the datasets listed in Table 8).

Moreover, Figure 14 demonstrates that autoencoder-based and adversarial approaches have made the most use of social datasets. Almost all of these autoencoder-based DGGs evaluate their generative models towards tasks such as link prediction and node classification instead of graph generation itself. In this respect, they differ from most of the reviewed methods, which aim to generate graphs from a similar distribution as the training data and perform the model evaluation using distribution-related metrics. Furthermore, nearly all of the adversarial approaches that have utilized social datasets to evaluate their models are trained to generate random walks instead of the whole graph. These indicate that current methods are not yet scalable enough to be easily used for generating large graphs such as social networks. Finally, Figure 14 unveils that utilizing synthetic datasets is more common between autoregressive, autoencoder-based, and flow-based DGGs. This kind of evaluation assesses the ability of generative models to learn particular patterns in synthetic datasets and is much more prevalent among non-molecular graph generators. Therefore, it is not welcomed by the RL-based approaches proposed so far, which are mainly designed for generating molecular graphs. This is also the case for the current adversarial DGGs as they mostly focus on generating molecular graphs or random walks of social networks.

B. EVALUATION METRICS

Depending on the application, graph generation approaches use different evaluation metrics. Specifically, the molecular graph generators adopt two different sets of metrics where the first set contains those evaluating the overall quality of generated samples, including *validity*, *uniqueness*, *novelty*, *reconstruction*, *internal diversity*, *negative log-likelihood (NLL)*, and some structural statistics like *nearest neighbor similarity (SNN)* or *fragment/scaffold similarity*. The second set, on the other hand, contains metrics assessing special chemical properties of the molecules, namely, *synthetic accessibility score (SA score)* [159], *drug-likeness score (QED)* [160], *molecular weight (MW)*, *log partition coefficient (logP)*, *penalized logP*, and *topological polar surface area (TPSA)*. For non-molecular graph generation, on the other side, a considerable number of approaches employ distribution-related metrics such as *Kullback-Leibler Divergence (KLD)* or *Maximum Mean Discrepancy (MMD)* on several graph statistics like degrees, clustering coefficients, orbit counts, and the spectra of the graphs from the eigenvalues of the normalized graph Laplacian. *NLL*, *validity*, *novelty*, and *uniqueness* are also among other metrics adopted to evaluate non-molecular approaches.

XI. CURRENT TRENDS AND FUTURE DIRECTIONS

In this section, we first discuss current trends of techniques and applications of deep graph generators and then suggest some potential future research directions.

A. CURRENT TRENDS

Due to the recent rise of deep learning-based graph generation methods, we are gradually witnessing the formation of some trends, revealing new research orientations. These trends include both the employed generation techniques and the applications for which these generators can be used. In this regard, Figure 15 demonstrates the utilization of the five categories' techniques by the most prominent reviewed DGGs during the past three years. This chart indicates a notable increase in the number of proposed autoregressive methods. It also unveils the inclination of researchers towards

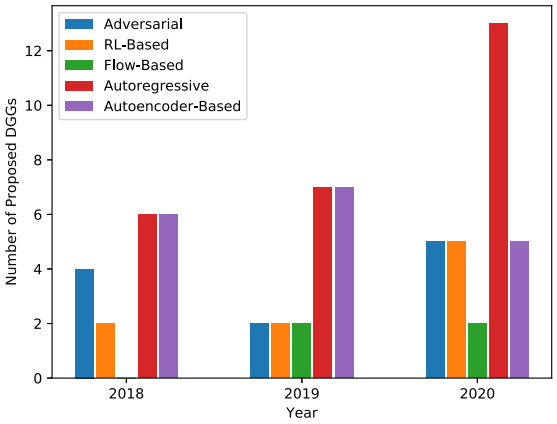

FIGURE 15. The number of the proposed DGGs (based on the methods referenced in Table 1) over time for each category.

applying flow-based techniques as a new approach for the graph generation task.

Moreover, Figure 16 shows that molecular applications have been important for modern graph generators and they still retain their importance in this field of research. Besides, due to the growing need for efficient design of drug molecules, it seems that employing DGGs to generate molecular graphs will continue to attract the attention of many researchers in the coming years. On the other hand, DGGs were initially less used in non-molecular applications, but each year more and more of them are designed to generate non-molecular graphs. One reason is the fact that these approaches have recently taken more steps towards scalability with the increase in the number of proposed autoregressive generators as discussed in Section VIII. However, in order to apply DGGs in generating more substantial non-molecular graphs such as social networks, which usually contain thousands of nodes, these methods need to be even more scalable. We will further discuss this issue as a future research direction in the following section.

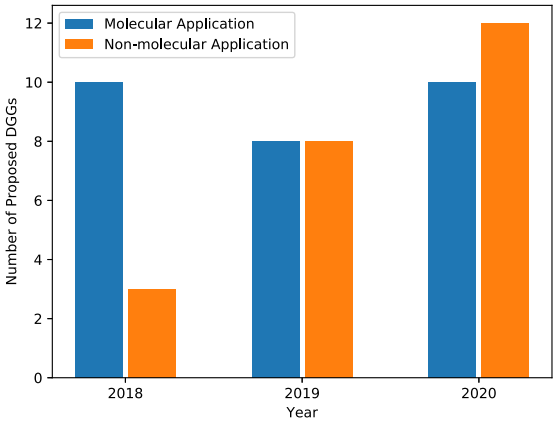

FIGURE 16. The number of the proposed DGGs (based on the methods referenced in Table 1) over time utilized in molecular and non-molecular applications.

TABLE 9. Acronyms with their extended names.

Acronym	Model Name	Reference
DGG	Deep Graph Generator	
RNN	Recurrent Neural Network	
LSTM	Long Short-Term Memory	[86]
GRU	Gated Recurrent Unit	[87]
VAE	Variational Autoencoder	[97]
GAN	Generative Adversarial Network	[64]
GNN	Graph Neural Network	[98]
GCN	Graph Convolutional Network	[1]
REIN	Recurrent Edge Inference Network	[91]
DeepNC	Deep Generative Network Completion	[92]
GRAN	Graph Recurrent Attention Network	[30]
GRAM	Graph Generative Model with Graph Attention Mechanism	[31]
AGE	Attention-Based Graph Evolution	[32]
DeepGMG	Deep Generative Models of Graphs	[33]
DeepGG	Deep Graph Generators	[94]
BiGG	BIg Graph Generation	[34]
VGAE	Variational Graph Autoencoder	[50]
MPGVAE	Message Passing Graph VAE	[52]
NED-VAE	Node-Edge Disentangled VAE	[55]
DGVAE	Dirichlet Graph VAE	[56]
JT-VAE	Junction Tree VAE	[57]
HierVAE	Hierarchical VAE	[37]
MHG-VAE	Molecular Hypergraph Grammar VAE	[58]
CGVAE	Constrained Graph VAE	[39]
DEFactor	Differentiable Edge Factorization-based Probabilistic Graph Generation	[40]
GraphVRNN	Graph Variational RNN	[41]
GCPN	Graph Convolutional Policy Network	[43]
MNCE-RL	Molecular Neighborhood-Controlled Embedding RL	[48]
GEGL	Genetic Expert-Guided Learning	[49]
MMGAN	Multi-MotifGAN	[66]
MolGAN	Molecular GAN	[63]
LGGAN	Labeled Graph GAN	[123]
TSGG-GAN	Time Series Conditioned Graph Generation-GAN	[68]
VJTNN	Variational Junction Tree Encoder-Decoder	[62]
Misc-GAN	Multi-Scale GAN	[70]
GNF	Graph Normalizing Flow	[60]
GrAD	Graph Auto-Decoder	[36]

B. FUTURE DIRECTIONS

Although several deep graph generation models have been proposed in the past few years, due to the emergence of this field and its short history, a number of open problems remain. In this subsection, considering both the aforementioned trends in the deep graph generation area and the current inclination of deep learning research, we suggest the following future directions.

TABLE 10. A set of publicly available source codes. O.A. = Original Authors.

Category	Method	URL	Language/Framework	O.A.
Autoregressive	MolMP [24]	https://github.com/kevinid/molecule_generator	Python/MXNet	Yes
	MolRNN [24]	https://github.com/kevinid/molecule_generator	Python/MXNet	Yes
	GraphRNN [25]	https://github.com/JiaxuanYou/graph-generation	Python/PyTorch	Yes
	DeepNC [92]	https://github.com/congasix/DeepNC	Python/TensorFlow	Yes
	Bacciu et al. [28]	https://github.com/marcopodda/grapher	Python/PyTorch	Yes
	GraphGen [29]	https://github.com/idea-iitd/graphgen	Python/PyTorch	Yes
	GRAN [30]	https://github.com/lrjconan/GRAN	Python/PyTorch	Yes
	DeepGMG [33]	https://github.com/JiaxuanYou/graph-generation/blob/master/main_DeepGMG.py	Python/PyTorch	No
	BiGG [34]	https://github.com/google-research/google-research/tree/master/bigg	Python/PyTorch	Yes
Autoencoder-Based	VGAE [50]	https://github.com/tkipf/gae	Python/TensorFlow	Yes
	GraphVAE [51]	https://github.com/JiaxuanYou/graph-generation/tree/master/baselines/graphvae	Python/PyTorch	No
	Graphite [54]	https://github.com/ermongroup/graphite	Python/TensorFlow	Yes
	NED-VAE [55]	https://github.com/xguo7/NED-VAE		Yes
	DGVAE [56]	https://github.com/xiyou3368/DGVAE	Python/TensorFlow	Yes
	JT-VAE [57]	https://github.com/wengong-jin/icml18-jtnn	Python/PyTorch	Yes
	HierVAE [37]	https://github.com/wengong-jin/hgraph2graph	Python/PyTorch	Yes
	MHG-VAE [58]	https://github.com/ibm-research-tokyo/graph_grammar	Python/PyTorch	Yes
	MoleculeChef [38]	https://github.com/john-bradshaw/molecule-chef	Python/PyTorch	Yes
	CGVAE [39]	https://github.com/microsoft/constrained-graph-variational-autoencoder	Python/TensorFlow	Yes
	NeVAE [59]	https://github.com/Networks-Learning/nevae	Python/TensorFlow	Yes
	Lim et al. [42]	https://github.com/jaechanglim/GGM		Yes
RL-Based	GCPN [43]	https://github.com/bowenliu16/rl_graph_generation	Python/TensorFlow	Yes
	Karimi et al. [45]	https://github.com/Shen-Lab/Drug-Combo-Generator	Python/TensorFlow	Yes
	DeepGraphMolGen [46]	https://github.com/dbkgroup/prop_gen	Python/PyTorch	Yes
	MNCE-RL [48]	https://github.com/Zoesgithub/MNCE-RL	Python/PyTorch	Yes
	GEGL [49]	https://github.com/sungsoo-ahn/genetic-expert-guided-learning	Python/PyTorch	Yes
Adversarial	NetGAN [65]	https://github.com/danielzuegner/netgan	Python/TensorFlow	Yes
	MolGAN [63]	https://github.com/nicola-decao/MolGAN	Python/TensorFlow	Yes
	CONDGEN [61]	https://github.com/KelestZ/CondGen	Python/PyTorch	Yes
	VJTNN + GAN [62]	https://github.com/wengong-jin/iclr19-graph2graph	Python/PyTorch	Yes
	Mol-CycleGAN [69]	https://github.com/ardigen/mol-cycle-gan	Python/Keras	Yes
	Misc-GAN [70]	https://github.com/Leo02016/Miscgan	Python/TensorFlow, Matlab	Yes
Flow-Based	GNF [60]	https://github.com/jliu/graph-normalizing-flows	Python/TensorFlow	Yes
	GraphNVP [71]	https://github.com/Kaushalya/graph-nvp	Python/Chainer-Chemistry	Yes
	GraphAF [35]	https://github.com/DeepGraphLearning/GraphAF	Python/PyTorch	Yes

1) SCALABILITY

Most of the proposed graph generation methods are only applicable to small graphs with a maximum of a few tens of nodes. Therefore, designing molecular graphs, which are mostly small in size, is the most prominent application of DGGs so far. Although some initial steps [25], [30], [34] have been taken towards scalability of generator models, much more effort is necessary to make them applicable in a wider range of real-world applications, such as social network modeling.

2) NODE ORDERING

Each graph with n nodes can be represented under $n!$ different node orderings. Therefore, it becomes intractable for

likelihood-based DGGs to calculate the exact likelihood as the graph size increases. To address this issue, some approximate approaches such as using approximate graph matching algorithms [51] or maximizing a lower bound of the likelihood by only considering subsets of node orderings (i.e., fixed [33], [34], uniform random [33], BFS [25], or a family of canonical orderings [30]), have been proposed. However, it is still necessary to provide more effective solutions for the node ordering problem, as a result of which, the generators can generate larger samples with higher quality.

3) INTERPRETABILITY

As mentioned earlier, DGGs are utilized in critical applications such as designing drug molecules, which directly affects public health. Therefore, the more transparent the generation procedure, the better control is exercised on the desirability of generated samples, which prevents additional trials and errors by limiting the number of candidate solutions. Hence, it is of great importance to make graph generation methods more interpretable. As of now, deep generative methods in areas such as image [161]–[163] and text [164]–[166] have slowly moved towards being more interpretable. However, only a few attempts [55], [56], [102] have recently been made in graph generation, making model interpretability a notable future research prospect.

4) DYNAMIC GRAPHS

While existing DGGs focus on generating static graphs, most of the graphs are inherently dynamic, meaning that they change over time by earning/losing nodes or connections, or even their attributes may alter. For example, in a social network, some users may join/leave the network, or the relationships between existing users may change over time. Therefore, generating dynamic graphs would play a key role in predicting how networks evolve. However, dynamicity is almost not addressed in the current generative approaches, making it a potentially challenging problem to explore in the further.

5) CONDITIONAL GRAPH GENERATION

When generating new graphs, in most cases, one aims to discover structures with desired characteristics. While conditional generation is relatively well investigated in image [19], [125], [167], [168] and text [169]–[171] domains, it is comparably less explored in the field of graph generation. For example, in molecular graph generation, the generated molecules must satisfy some validity constraints or hold desired chemical properties. However, only a limited number of proposed methods adopt a conditional approach by whether incorporating conditional codes into the generation process [42], [51], enforcing the existence of favourable substructures in the output graph [42] or performing the generation conditioned on an input molecule to ensure the structural similarity [37]. Meanwhile, the majority of methods do not formulate the problem as conditional generation and address the issue by employing other techniques like property optimization in some latent continuous space [38], [39], [57]–[59] or injecting validity constraints, whether at the training [26], [53] or the inference time [26], [35]. Nevertheless, this issue has been even less studied in the non-molecular graph generation models, and only a few of them have partially addressed the problem [61], [67]. Therefore, focusing more on conditional graph generation problems, especially those that have not yet been explored, such as class conditioned generation, is an important future research direction.

XII. CONCLUSION

In this article, we surveyed the emerging field of deep learning-based graph generation. For this purpose, we classified the existing methods into five general categories. We then provided a detailed and comparative review of the approaches in each category. We further summarized the categories' characteristics and compared them from multiple points of view. Next, we summarized the implementation details, including datasets, evaluation metrics, and available source codes, and discussed the current applications and possible future ones. Finally, we reviewed current trends both in techniques and applications and suggested future research directions according to the current trends and challenges. We believe this article provides the readers a comprehensive insight to the field of graph generation research.

APPENDIX A
ACRONYM

Table 9 summarizes the acronyms and nomenclature used in this survey.

APPENDIX B
SOURCE CODES

Table 10 summarizes the set of publicly available source codes for deep learning-based graph generation approaches discussed in the survey.

REFERENCES

[1] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. Int. Conf. Learn. Represent.*, 2017, pp. 1–14.
[2] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 1024–1034.
[3] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–17.
[4] L. Cotta, C. H. Teixeira, A. Swami, and B. Ribeiro, "Unsupervised joint k-node graph representations with compositional energy-based models," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 1–20.
[5] M. Ramezani, W. Cong, M. Mahdavi, A. Sivasubramaniam, and A. T. Kandemir, "GCN meets GPU: Decoupling 'when to sample' from 'how to sample,'" in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 1–27.
[6] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *IEEE Data Eng. Bull.*, vol. 40, no. 3, pp. 52–74, Sep. 2017.
[7] Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli, "Graph matching networks for learning the similarity of graph structured objects," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 3835–3845.

[8] M. Fey, J. E. Lenssen, C. Morris, J. Masci, and N. M. Kriege, "Deep graph matching consensus," in *Proc. Int. Conf. Learn. Represent.*, 2020, pp. 1–23.
[9] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2018, pp. 2847–2856.
[10] D. Zügner and S. Günnemann, "Adversarial attacks on graph neural networks via meta learning," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–15.
[11] P. Veli ković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–12.
[12] V. Kosaraju, A. Sadeghian, R. Martín-Martín, I. Reid, H. Rezatofighi, and S. Savarese, "Social-BiGAT: Multimodal trajectory forecasting using bicycle-GAN and graph attention networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 137–146.
[13] P. Erdős and A. Rényi, "On the evolution of random graphs," *Publications Math. Inst. Hung. Acad. Sci.*, vol. 5, no. 1, pp. 17–60, 1960.
[14] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.
[15] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Rev. Mod. Phys.*, vol. 74, p. 47, Jan. 2002.
[16] P. W. Holland, K. B. Laskey, and S. Leinhardt, "Stochastic blockmodels: First steps," *Social Netw.*, vol. 5, no. 2, pp. 109–137, 1983.
[17] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and A. Ghahramani, "Kronecker graphs: An approach to modeling networks," *J. Mach. Learn. Res.*, vol. 11, no. 2, pp. 1–58, 2010.
[18] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," in *Proc. Int. Conf. Learn. Represent.*, 2016, pp. 1–16.
[19] X. Yan, J. Yang, K. Sohn, and H. Lee, "Attribute2image: Conditional image generation from visual attributes," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 776–791.
[20] Y. Zhang, Z. Gan, K. Fan, Z. Chen, R. Henao, D. Shen, and A. Carin, "Adversarial feature matching for text generation," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 4006–4015.
[21] P. Z. Wang and W. Y. Wang, "Riemannian normalizing flow on variational Wasserstein autoencoder for text modeling," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol. (NAACL-HLT)*, vol. 1. Minneapolis, MN, USA: Association for Computational Linguistics, Jun. 2019, pp. 284–294.
[22] T. Kaneko, H. Kameoka, K. Hiramatsu, and K. Kashino, "Sequence-to-sequence voice conversion with similarity metric learned using generative adversarial networks," in *Proc. INTERSPEECH*, Aug. 2017, pp. 1283–1287.
[23] Y. Gao, R. Singh, and B. Raj, "Voice impersonation using generative adversarial networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2018, pp. 2506–2510.
[24] Y. Li, L. Zhang, and Z. Liu, "Multi-objective de novo drug design with conditional graph generative model," *J. Cheminformatics*, vol. 10, no. 1, p. 33, Dec. 2018.
[25] J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec, "GraphRNN: Generating realistic graphs with deep auto-regressive models," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 5708–5717.
[26] M. Popova, M. Shvets, J. Oliva, and O. Isayev, "MolecularRNN: Generating realistic molecular graphs with optimized properties," 2019, *arXiv:1905.13372*. [Online]. Available: http://arxiv.org/abs/1905.13372
[27] D. Bacciu, A. Micheli, and M. Podda, "Graph generation by sequential edge prediction," in *Proc. ESANN*, 2019, pp. 95–100.
[28] D. Bacciu, A. Micheli, and M. Podda, "Edge-based sequential graph generation with recurrent neural networks," *Neurocomputing*, vol. 416, pp. 177–189, Nov. 2020.
[29] N. Goyal, H. V. Jain, and S. Ranu, "GraphGen: A scalable approach to domain-agnostic labeled graph generation," in *Proc. Web Conf.*, Apr. 2020, pp. 1253–1263.
[30] R. Liao, Y. Li, Y. Song, S. Wang, W. Hamilton, D. K. Duvenaud, R. Urtasun, and R. Zemel, "Efficient graph generation with graph recurrent attention networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 4257–4267.
[31] W. Kawai, Y. Mukuta, and T. Harada, "Scalable generative models for graphs with graph attention mechanism," 2019, *arXiv:1906.01861*. [Online]. Available: http://arxiv.org/abs/1906.01861
[32] S. Fan and B. Huang, "Attention-based graph evolution," in *Proc. Pacific-Asia Conf. Knowl. Discovery Data Mining*. Cham, Switzerland: Springer, 2020, pp. 436–447.
[33] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia, "Learning deep generative models of graphs," 2018, *arXiv:1803.03324*. [Online]. Available: http://arxiv.org/abs/1803.03324
[34] H. Dai, A. Nazi, Y. Li, B. Dai, and D. Schuurmans, "Scalable deep generative modeling for sparse graphs," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 2302–2312.
[35] C. Shi, M. Xu, Z. Zhu, W. Zhang, M. Zhang, and J. Tang, "GraphAF: A flow-based autoregressive model for molecular graph generation," in *Proc. Int. Conf. Learn. Represent.*, 2020, pp. 1–18.
[36] S. A. Shah and V. Koltun, "Auto-decoding graphs," 2020, *arXiv:2006.02879*. [Online]. Available: http://arxiv.org/abs/2006.02879
[37] W. Jin, R. Barzilay, and T. Jaakkola, "Hierarchical generation of molecular graphs using structural motifs," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 4839–4848.
[38] J. Bradshaw, B. Paige, M. J. Kusner, M. Segler, and J. M. Hernández-Lobato, "A model to search for synthesizable molecules," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 7935–7947.
[39] Q. Liu, M. Allamanis, M. Brockschmidt, and A. Gaunt, "Constrained graph variational autoencoders for molecule design," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 7795–7804.
[40] R. Assouel, M. Ahmed, M. H Segler, A. Saffari, and Y. Bengio, "DEFactor: Differentiable edge factorization-based probabilistic graph generation," 2018, *arXiv:1811.09766*. [Online]. Available: http://arxiv.org/abs/1811.09766
[41] S.-Y. Su, H. Hajimirsadeghi, and G. Mori, "Graph generation with variational recurrent neural network," in *Proc. NIPS Workshop Graph Represent. Learn.*, 2019, pp. 1–8.
[42] J. Lim, S.-Y. Hwang, S. Moon, S. Kim, and W. Y. Kim, "Scaffold-based molecular design with a graph generative model," *Chem. Sci.*, vol. 11, no. 4, pp. 1153–1164, 2020.
[43] J. You, B. Liu, Z. Ying, V. Pande, and J. Leskovec, "Graph convolutional policy network for goal-directed molecular graph generation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 6410–6421.
[44] F. Shi, S. You, and C. Xu, "Reinforced molecule generation with heterogeneous states," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, Nov. 2019, pp. 548–557.
[45] M. Karimi, A. Hasanzadeh, and Y. Shen, "Network-principled deep generative models for designing drug combinations as graph sets," *Bioinformatics*, vol. 36, no. 1, pp. i445–i454, Jul. 2020.
[46] Y. Khemchandani, S. O'Hagan, S. Samanta, N. Swainston, T. J. Roberts, D. Bollegala, and D. B. Kell, "DeepGraphMolGen, a multi-objective, computational strategy for generating molecules with desirable properties: A graph convolution and reinforcement learning approach," *J. Cheminformatics*, vol. 12, no. 1, pp. 1–17, Dec. 2020.
[47] R. Trivedi, J. Yang, and H. Zha, "GraphOpt: Learning optimization models of graph formation," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 9603–9613.
[48] C. Xu, Q. Liu, M. Huang, and T. Jiang, "Reinforced molecular optimization with neighborhood-controlled grammars," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 1–25.
[49] S. Ahn, J. Kim, H. Lee, and J. Shin, "Guiding deep molecular optimization with genetic exploration," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 1–20.
[50] T. N. Kipf and M. Welling, "Variational graph auto-encoders," in *Proc. NIPS Workshop Bayesian Deep Learn.*, 2016, pp. 1–3.
[51] M. Simonovsky and N. Komodakis, "GraphVAE: Towards generation of small graphs using variational autoencoders," in *Proc. Int. Conf. Artif. Neural Netw.* Cham, Switzerland: Springer, 2018, pp. 412–422.
[52] D. Flam-Shepherd, T. Wu, and A. Aspuru-Guzik, "Graph deconvolutional generation," 2020, *arXiv:2002.07087*. [Online]. Available: http://arxiv.org/abs/2002.07087
[53] T. Ma, J. Chen, and C. Xiao, "Constrained generation of semantically valid graphs via regularizing variational autoencoders," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 7113–7124.
[54] A. Grover, A. Zweig, and S. Ermon, "Graphite: Iterative generative modeling of graphs," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 2434–2444.
[55] X. Guo, L. Zhao, Z. Qin, L. Wu, A. Shehu, and Y. Ye, "Node-edge co-disentangled representation learning for attributed graph generation," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2020, pp. 1–10.
[56] J. Li, J. Yu, J. Li, H. Zhang, K. Zhao, Y. Rong, H. Cheng, and J. Huang, "Dirichlet graph variational autoencoder," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 1–13.

[57] W. Jin, R. Barzilay, and T. Jaakkola, "Junction tree variational autoencoder for molecular graph generation," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 2323–2332.
[58] H. Kajino, "Molecular hypergraph grammar with its application to molecular optimization," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 3183–3191.
[59] B. Samanta, D. Abir, G. Jana, P. K. Chattaraj, N. Ganguly, and A. M. G. Rodriguez, "NEVAE: A deep generative model for molecular graphs," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 1110–1117.
[60] J. Liu, A. Kumar, J. Ba, J. Kiros, and K. Swersky, "Graph normalizing flows," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 13578–13588.
[61] C. Yang, P. Zhuang, W. Shi, A. Luu, and P. Li, "Conditional structure generation through graph variational generative adversarial nets," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 1340–1351.
[62] W. Jin, K. Yang, R. Barzilay, and T. Jaakkola, "Learning multimodal graph-to-graph translation for molecule optimization," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–13.
[63] N. D. Cao and T. Kipf, "MolGAN: An implicit generative model for small molecular graphs," in *Proc. ICML Workshop Theor. Found. Appl. Deep Generative Models*, 2018, pp. 1–11.
[64] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2672–2680.
[65] A. Bojchevski, O. Shchur, D. Zügner, and S. Günnemann, "NetGAN: Generating graphs via random walks," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 609–618.
[66] A. Gamage, E. Chien, J. Peng, and O. Milenkovic, "Multi-MotifGAN (MMGAN): Motif-targeted graph generation and prediction," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2020, pp. 4182–4186.
[67] W. Joon-Wie Tann, E.-C. Chang, and B. Hooi, "SHADOWCAST: Controllable graph generation," 2020, *arXiv:2006.03774*. [Online]. Available: http://arxiv.org/abs/2006.03774
[68] S. Yang, J. Liu, K. Wu, and M. Li, "Learn to generate time series conditioned graphs with generative adversarial nets," 2020, *arXiv:2003.01436*. [Online]. Available: http://arxiv.org/abs/2003.01436
[69] Ł. Maziarka, A. Pocha, J. Kaczmarczyk, K. Rataj, T. Danel, and M. Warchoł, "Mol-CycleGAN: A generative model for molecular optimization," *J. Cheminformatics*, vol. 12, no. 1, pp. 1–18, Dec. 2020.
[70] D. Zhou, L. Zheng, J. Xu, and J. He, "Misc-GAN: A multi-scale generative model for graphs," *Frontiers Big Data*, vol. 2, p. 3, Apr. 2019.
[71] K. Madhawa, K. Ishiguro, K. Nakago, and M. Abe, "GraphNVP: An invertible flow model for generating molecular graphs," 2019, *arXiv:1905.11600*. [Online]. Available: http://arxiv.org/abs/1905.11600
[72] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *IEEE Trans. Knowl. Data Eng.*, early access, Mar. 17, 2020, doi: 10.1109/TKDE.2020.2981333.
[73] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 4–24, Jan. 2021.
[74] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," 2018, *arXiv:1812.08434*. [Online]. Available: http://arxiv.org/abs/1812.08434
[75] W. Cao, Z. Yan, Z. He, and Z. He, "A comprehensive survey on geometric deep learning," *IEEE Access*, vol. 8, pp. 35929–35949, 2020.
[76] D. Bacciu, F. Errica, A. Micheli, and M. Podda, "A gentle introduction to deep learning for graphs," *Neural Netw.*, vol. 129, pp. 203–221, Sep. 2020.
[77] K. Sun, L. Wang, B. Xu, W. Zhao, S. W. Teng, and F. Xia, "Network representation learning: From traditional feature learning to deep learning," *IEEE Access*, vol. 8, pp. 205600–205617, 2020.
[78] S. Georgousis, M. P. Kenning, and X. Xie, "Graph deep learning: State of the art and challenges," *IEEE Access*, vol. 9, pp. 22106–22140, 2021.
[79] J. B. Lee, R. A. Rossi, S. Kim, N. K. Ahmed, and E. Koh, "Attention models in graphs: A survey," *ACM Trans. Knowl. Discovery Data*, vol. 13, no. 6, pp. 1–25, 2019.
[80] F. Lu, P. Cong, and X. Huang, "Utilizing textual information in knowledge graph embedding: A survey of methods and applications," *IEEE Access*, vol. 8, pp. 92072–92088, 2020.
[81] S. Ji, S. Pan, E. Cambria, P. Marttinen, and P. S. Yu, "A survey on knowledge graphs: Representation, acquisition, and applications," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Apr. 26, 2021, doi: 10.1109/TNNLS.2021.3070843.
[82] L. Sun, Y. Dou, C. Yang, J. Wang, P. S. Yu, L. He, and B. Li, "Adversarial attack and defense on graph data: A survey," 2018, *arXiv:1812.10528*. [Online]. Available: http://arxiv.org/abs/1812.10528
[83] G. Ma, N. K. Ahmed, T. L. Willke, and P. S. Yu, "Deep graph similarity learning: A survey," 2019, *arXiv:1912.11615*. [Online]. Available: http://arxiv.org/abs/1912.11615
[84] J. Yan, S. Yang, and E. Hancock, "Learning for graph matching and related combinatorial optimization problems," in *Proc. Int. Joint Conf. Artif. Intell.*, York, U.K., 2020, pp. 1–10.
[85] X. Guo and L. Zhao, "A systematic survey on deep generative models for graph generation," 2020, *arXiv:2007.06686*. [Online]. Available: http://arxiv.org/abs/2007.06686
[86] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[87] K. Cho, B. V. Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1724–1734.
[88] C.-C. Liu, H. Chan, K. Luk, and A. I. Borealis, "Auto-regressive graph generation modeling with improved evaluation methods," in *Proc. NeurIPS Workshop Graph Represent. Learn.*, 2019. [Online]. Available: https://grlearning.github.io/papers/77.pdf
[89] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.
[90] 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.
[91] R. Daroya, R. Atienza, and R. Cajote, "REIN: Flexible mesh generation from point clouds," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2020, pp. 352–353.
[92] C. Tran, W.-Y. Shin, A. Spitz, and M. Gertz, "DeepNC: Deep generative network completion," *IEEE Trans. Pattern Anal. Mach. Intell.*, early access, Oct. 19, 2020, doi: 10.1109/TPAMI.2020.3032286.
[93] X. Yan and J. Han, "GSpan: Graph-based substructure pattern mining," in *Proc. IEEE Int. Conf. Data Mining*, Dec. 2002, pp. 721–724.
[94] J. Stier and M. Granitzer, "DeepGG: A deep graph generator," 2020, *arXiv:2006.04159*. [Online]. Available: http://arxiv.org/abs/2006.04159
[95] D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-MAT: A recursive model for graph mining," in *Proc. SIAM Int. Conf. Data Mining*, Apr. 2004, pp. 442–446.
[96] P. M. Fenwick, "A new data structure for cumulative frequency tables," *Softw., Pract. Exper.*, vol. 24, no. 3, pp. 327–336, Mar. 1994.
[97] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," in *Proc. Int. Conf. Learn. Represent.*, 2014, pp. 1–14.
[98] F. Scarselli, M. Gori, 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.
[99] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 3693–3702.
[100] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, 2017, pp. 1263–1272.
[101] O. Vinyals, S. Bengio, and M. Kudlur, "Order matters: Sequence to sequence for sets," 2015, *arXiv:1511.06391*. [Online]. Available: http://arxiv.org/abs/1511.06391
[102] N. Stoehr, M. Brockschmidt, J. Stuehmer, and E. Yilmaz, "Disentangling interpretable generative parameters of random and real-world graphs," in *Proc. NIPS Workshop Graph Represent. Learn.*, 2019, pp. 1–8.
[103] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "β-VAE: Learning basic visual concepts with a constrained variational framework," in *Proc. Int. Conf. Learn. Represent.*, 2017, pp. 1–22.
[104] P. Hennig, D. Stern, R. Herbrich, and T. Graepel, "Kernel topic models," in *Proc. Artif. Intell. Statist.*, 2012, pp. 511–519.
[105] F. Drewes, H.-J. Kreowski, and A. Habel, "Hyperedge replacement graph grammars," in *Handbook Of Graph Grammars And Computing By Graph Transformation: Foundations*, vol. 1. Singapore: World Scientific, 1997, pp. 95–162.
[106] M. Kusner, B. Paige, and J. Hernández-Lobato, "Grammar variational autoencoder," in *Proc. 34th Int. Conf. Mach. Learn.*, Sydney, NSW, Australia, vol. 70, 2017, pp. 1945–1954.
[107] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," in *Proc. Int. Conf. Learn. Represent.*, 2016, pp. 1–20.

[108] P. Schwaller, T. Laino, T. Gaudin, P. Bolgar, C. A. Hunter, C. Bekas, and A. A. Lee, "Molecular transformer: A model for uncertainty-calibrated chemical reaction prediction," *ACS Central Sci.*, vol. 5, no. 9, pp. 1572–1583, Sep. 2019.
[109] I. Tolstikhin, O. Bousquet, S. Gelly, and B. Schölkopf, "Wasserstein auto-encoders," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–20.
[110] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 3483–3491.
[111] P. W. Battaglia, R. Pascanu, M. Lai, D. Rezende, and K. Kavukcuoglu, "Interaction networks for learning about objects, relations and physics," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 4502–4510.
[112] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*. [Online]. Available: http://arxiv.org/abs/1707.06347
[113] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning," in *Proc. ICML*, vol. 1, 2000, p. 2.
[114] C. Finn, S. Levine, and P. Abbeel, "Guided cost learning: Deep inverse optimal control via policy optimization," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 49–58.
[115] D. Janssens and G. Rozenberg, "Graph grammars with neighbourhood-controlled embedding," *Theor. Comput. Sci.*, vol. 21, no. 1, pp. 55–74, Oct. 1982.
[116] J. H. Jensen, "A graph-based genetic algorithm and generative model/Monte Carlo tree search for the exploration of chemical space," *Chem. Sci.*, vol. 10, no. 12, pp. 3567–3572, 2019.
[117] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, 2017, pp. 214–223.
[118] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 855–864.
[119] L. Zhang, "STGGAN: Spatial-temporal graph generation," in *Proc. 27th ACM SIGSPATIAL Int. Conf. Adv. Geograph. Inf. Syst.*, Nov. 2019, pp. 608–609.
[120] M. Mirza and S. Osindero, "Conditional generative adversarial nets," in *Proc. NIPS Workshop Deep Learn. Represent.*, 2014, pp. 1–7.
[121] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *Proc. Eur. Semantic Web Conf.* Cham, Switzerland: Springer, 2018, pp. 593–607.
[122] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, 2016, pp. 1–14.
[123] S. Fan and B. Huang, "Conditional labeled graph generation with GANs," in *Proc. ICLR Workshop Represent. Learn. Graphs Manifolds*, 2019. [Online]. Available: https://rlgm.github.io/papers/22.pdf
[124] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-I. Kawarabayashi, and A. S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 5453–5462.
[125] A. Odena, C. Olah, and J. Shlens, "Conditional image synthesis with auxiliary classifier GANs," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 2642–2651.
[126] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 2223–2232.
[127] T. Lei, Y. Zhang, S. I. Wang, H. Dai, and Y. Artzi, "Simple recurrent units for highly parallelizable recurrence," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2018, pp. 4470–4481.
[128] R. Socher, D. Chen, C. D. Manning, and A. Ng, "Reasoning with neural tensor networks for knowledge base completion," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 926–934.
[129] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang, and S. P. Smolley, "Least squares generative adversarial networks," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 2794–2802.
[130] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real NVP," in *Proc. Int. Conf. Learn. Represent.*, 2017, pp. 1–32.
[131] D. P. Kingma and P. Dhariwal, "Glow: Generative flow with invertible 1×1 convolutions," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 10215–10224.
[132] G. Papamakarios, T. Pavlakou, and I. Murray, "Masked autoregressive flow for density estimation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 2338–2347.
[133] M. Olivecrona, T. Blaschke, O. Engkvist, and H. Chen, "Molecular de-novo design through deep reinforcement learning," *J. Cheminformatics*, vol. 9, no. 1, p. 48, Dec. 2017.
[134] H. Dai, Y. Tian, B. Dai, S. Skiena, and L. Song, "Syntax-directed variational autoencoder for structured data," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–17.
[135] R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik, "Automatic chemical design using a data-driven continuous representation of molecules," *ACS Central Sci.*, vol. 4, no. 2, pp. 268–276, 2018.
[136] M. Popova, O. Isayev, and A. Tropsha, "Deep reinforcement learning for de novo drug design," *Sci. Adv.*, vol. 4, no. 7, Jul. 2018, Art. no. eaap7885.
[137] C. Wang, N. Xue, and S. Pradhan, "A transition-based algorithm for AMR parsing," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.*, 2015, pp. 366–375.
[138] 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.
[139] S. Zhang, X. Ma, K. Duh, and B. Van Durme, "AMR parsing as sequence-to-graph transduction," in *Proc. 57th Annu. Meeting Assoc. Comput. Linguistics*, 2019, pp. 80–94.
[140] 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.
[141] T. Dozat and D. C. Manning, "Deep biaffine attention for neural dependency parsing," in *Proc. 5th Int. Conf. Learn. Represent. (ICLR)*, Toulon, France, Apr. 2017, pp. 1–8.
[142] Y. Wang, W. Che, J. Guo, and T. Liu, "A neural transition-based approach for semantic dependency graph parsing," in *Proc. AAAI*, 2018, pp. 5561–5568.
[143] T. Dozat and C. D. Manning, "Simpler but more accurate semantic dependency parsing," in *Proc. 56th Annu. Meeting Assoc. Comput. Linguistics*, vol. 2, 2018, pp. 484–490.
[144] D. D. Johnson, "Learning graphical state transitions," in *Proc. 5th Int. Conf. Learn. Represent. (ICLR)*, Toulon, France, Apr. 2017, pp. 1–19.
[145] J. Gu, H. Zhao, Z. Lin, S. Li, J. Cai, and M. Ling, "Scene graph generation with external knowledge and image reconstruction," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 1969–1978.
[146] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 91–99.
[147] Y. Li, W. Ouyang, B. Zhou, K. Wang, and X. Wang, "Scene graph generation from objects, phrases and region captions," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 1261–1270.
[148] D. Xu, Y. Zhu, C. B. Choy, and L. Fei-Fei, "Scene graph generation by iterative message passing," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 5410–5419.
[149] J. Yang, J. Lu, S. Lee, D. Batra, and D. Parikh, "Graph R-CNN for scene graph generation," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 670–685.
[150] M. Qi, W. Li, Z. Yang, Y. Wang, and J. Luo, "Attentive relational networks for mapping images to scene graphs," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 3957–3966.
[151] T. Chen, W. Yu, R. Chen, and L. Lin, "Knowledge-embedded routing network for scene graph generation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 6163–6171.
[152] A. Newell and J. Deng, "Pixels to graphs by associative embedding," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 2171–2180.
[153] M. Khademi and O. Schulte, "Deep generative probabilistic graph neural networks for scene graph generation," in *Proc. AAAI*, 2020, pp. 11237–11245.
[154] Y. Li, W. Ouyang, B. Zhou, J. Shi, C. Zhang, and X. Wang, "Factorizable net: An efficient subgraph-based framework for scene graph generation," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 335–351.
[155] M. Klawonn and E. Heim, "Generating triples with adversarial networks for scene graph construction," in *Proc. AAAI*, 2018, pp. 1–8.
[156] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 1–34.
[157] N. Brown, M. Fiscato, M. H. S. Segler, and A. C. Vaucher, "GuacaMol: Benchmarking models for de novo molecular design," *J. Chem. Inf. Model.*, vol. 59, no. 3, pp. 1096–1108, Mar. 2019.
[158] D. Polykovskiy, A. Zhebrak, B. Sanchez-Lengeling, S. Golovanov, O. Tatanov, S. Belyaev, R. Kurbanov, A. Artamonov, V. Aladinskiy, M. Veselov, A. Kadurin, S. Johansson, H. Chen, S. Nikolenko, A. Aspuru-Guzik, and A. Zhavoronkov, "Molecular sets (MOSES): A benchmarking platform for molecular generation models," *Frontiers Pharmacol.*, vol. 11, p. 1931, Dec. 2020.

[159] P. Ertl and A. Schuffenhauer, "Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions," *J. Cheminformatics*, vol. 1, no. 1, p. 8, 2009.
[160] G. R. Bickerton, G. V. Paolini, J. Besnard, S. Muresan, and A. L. Hopkins, "Quantifying the chemical beauty of drugs," *Nature Chem.*, vol. 4, no. 2, pp. 90–98, Feb. 2012.
[161] C. Biffi, O. Oktay, G. Tarroni, W. Bai, A. De Marvao, G. Doumou, M. Rajchl, R. Bedair, S. Prasad, S. Cook, D. O'Regan, and D. Rueckert, "Learning interpretable anatomical features through deep generative models: Application to cardiac remodeling," in *Proc. Int. Conf. Med. Image Comput. Comput.-Assist. Intervent.* Cham, Switzerland: Springer, 2018, pp. 464–471.
[162] A. Voynov and A. Babenko, "RPGAN: GANs interpretability via random routing," 2019, *arXiv:1912.10920*. [Online]. Available: http://arxiv.org/abs/1912.10920
[163] C. Biffi, J. J. Cerrolaza, G. Tarroni, W. Bai, A. de Marvao, O. Oktay, C. Ledig, L. Le Folgoc, K. Kamnitsas, G. Doumou, J. Duan, S. K. Prasad, S. A. Cook, D. P. O'Regan, and D. Rueckert, "Explainable anatomical shape analysis through deep hierarchical generative models," *IEEE Trans. Med. Imag.*, vol. 39, no. 6, pp. 2088–2099, Jun. 2020.
[164] T.-H. Wen, Y. Miao, P. Blunsom, and S. Young, "Latent intention dialogue models," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 3732–3741.
[165] T. Zhao, K. Lee, and M. Eskenazi, "Unsupervised discrete sentence representation learning for interpretable neural dialog generation," in *Proc. 56th Annu. Meeting Assoc. Comput. Linguistics*, 2018, pp. 1098–1107.
[166] W. Shi, H. Zhou, N. Miao, and L. Li, "Dispersed exponential family mixture VAEs for interpretable text generation," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 8840–8851.
[167] A. Van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, and A. Graves, "Conditional image generation with PixelCNN decoders," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 4790–4798.
[168] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz, and B. Catanzaro, "High-resolution image synthesis and semantic manipulation with conditional GANs," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8798–8807.
[169] Z. Hu, Z. Yang, X. Liang, R. Salakhutdinov, and E. P. Xing, "Toward controlled generation of text," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1587–1596.
[170] H. Zhou, M. Huang, T. Zhang, X. Zhu, and B. Liu, "Emotional chatting machine: Emotional conversation generation with internal and external memory," in *Proc. AAAI*, 2018, pp. 1–9.
[171] N. S. Keskar, B. McCann, L. R. Varshney, C. Xiong, and R. Socher, "CTRL: A conditional transformer language model for controllable generation," 2019, *arXiv:1909.05858*. [Online]. Available: http://arxiv.org/abs/1909.05858

FAEZEH FAEZ received the B.Sc. and M.Sc. degrees in software engineering from the Sharif University of Technology, Tehran, Iran, where she is currently pursuing the Ph.D. degree in artificial intelligence with the Department of Computer Engineering. Her current research interests include machine learning, deep learning, and deep graph generative models.

YASSAMAN OMMI is currently pursuing the B.Sc. degree in computer science with the Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran. Her current research interests include graph-based deep learning, pattern recognition, and complex networks.

MAHDIEH SOLEYMANI BAGHSHAH received the B.Sc., M.Sc., and Ph.D. degrees from the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran, in 2003, 2005, and 2010, respectively. She is currently an Assistant Professor with the Computer Engineering Department, Sharif University of Technology. Her research interests include machine learning and deep learning.

HAMID R. RABIEE (Senior Member, IEEE) received the B.S. and M.S. degrees (Hons.) in electrical engineering from CSULB, Long Beach, CA, USA, in 1987 and 1989, respectively, the E.E.E. degree in electrical and computer engineering from USC, Los Angeles, CA, USA, in 1993, and the Ph.D. degree in electrical and computer engineering from Purdue University, West Lafayette, IN, USA, in 1996. From 1993 to 1996, he was a Member of Technical Staff with AT&T Bell Laboratories. From 1996 to 1999, he worked as a Senior Software Engineer with Intel Corporation. He was with PSU, OGI, and OSU universities, as an Adjunct Professor of electrical and computer engineering, from 1996 to 2000. Since September 2000, he has been with the Sharif University of Technology, Tehran, Iran, where he is currently a Professor of computer engineering. He was a Visiting Professor with Imperial College London, for the 2017–2018 academic year. He is also the Founder of the Sharif University Advanced Information and Communication Technology Research Institute (AICT), the ICT Innovation Center, the Advanced Technologies Incubator (SATI), the Digital Media Laboratory (DML), the Mobile Value Added Services Laboratory (VASL), the Bioinformatics and Computational Biology Laboratory (BCB), and the Cognitive Neuroengineering Research Center. He is also a Consultant and a member of AI with the Health Expert Group, WHO. He has been the founder of many successful high-tech start-up companies in the field of ICT, as an entrepreneur. He is also the Director of AICT, DML, and VASL. He holds three patents. His research interests include statistical machine learning, Bayesian statistics, data analytics and complex networks with applications in social networks, multimedia systems, cloud, and the IoT privacy, bioinformatics, and brain networks. He received numerous awards and honors for his industrial, scientific, and academic contributions.