GRAM: Scalable Generative Models for Graphs with Graph Attention Mechanism

Wataru Kawai
The University of Tokyo
w-kawai@mi.t.u-tokyo.ac.jp

Yusuke Mukuta
The University of Tokyo, RIKEN
mukuta@mi.t.u-tokyo.ac.jp

Tatsuya Harada
The University of Tokyo, RIKEN
harada@mi.t.u-tokyo.ac.jp

Abstract

Graphs are ubiquitous real-world data structures, and generative models that can approximate distributions over graphs and derive samples from it have significant importance. There are several known challenges in graph generation tasks, and *scalability* handling large graphs and datasets is one of the most important for applications in a wide range of real-world domains. Although an increasing number of graph generative models have been proposed in the field of machine learning that have demonstrated impressive results in several tasks, scalability is still an unresolved problem owing to the complex generation process or difficulty in training parallelization. In this work, we first define scalability from three different perspectives: number of nodes, data, and node/edge labels, and then we propose GRAM, a generative model for real-world graphs that is scalable in all the three contexts, especially on training. We aim to achieve scalability by employing a novel graph attention mechanism, formulating the likelihood of graphs in a simple and general manner and utilizing the properties of real-world graphs such as community structure and sparseness of edges. Furthermore, we construct a non-domain-specific evaluation metric in node/edge-labeled graph generation tasks that combine a graph kernel and Maximum Mean Discrepancy. Our experiments on real-world graph datasets showed that our models can scale up to large graphs and datasets that baseline models had difficulty handling, and demonstrated results that were competitive with or superior than the baseline methods.

1 Introduction

Graphs are ubiquitous and fundamental data structures in the real world, appearing in various fields such as social science, chemistry, and biology. For example, knowledge bases, molecular structures, and relations between objects in an image can be represented as graphs, which allows us to efficiently capture the essence of these data. Moreover, in applications such as drug discovery or network simulation, graph generative models that can approximate distributions over graphs on a specific domain and derive new samples from it are very important. Compared with generation tasks for images or natural language, however, graph generation tasks are significantly difficult due to the necessity of modeling complex local/global dependencies between nodes and edges, as well as the intractable properties of graphs themselves, such as discreteness, variable number of nodes and edges, and uncertainty of node ordering.

Although graph generation tasks encompass several challenges as described above, *scalability* is one of the most important for applications in a wide range of real-world domains. In this work, we

Preprint. Under review.

define scalability from three perspectives: yielding *graph scalability*, *data scalability*, and *label scalability*. Graph scalability denotes scalability to large graphs with many nodes, data scalability denotes scalability to large datasets with many data, and label scalability denotes scalability to graphs with many node/edge labels within the limit of practical time/space complexity, especially on training. In addition to these perspectives, it is possible to consider other viewpoints, such as the number of edges, diameter of graphs, or total number of nodes in datasets. However, because these perspectives are closely related to the above three points, we adopt these three as a mutually exclusive and collectively exhaustive division.

In recent years, an increasing number of graph generative models based on machine learning have been proposed, which have demonstrated great performances in several tasks, such as link prediction, molecular property optimization, and network structure optimization [20, 31, 13, 23, 30, 22, 25, 29, 21]. However, to the best of our knowledge, no model is scalable in all three contexts. For example, DeepGMG [20] can generate only small graphs with fewer than 40 nodes, GraphRNN [31] cannot generate node/edge-labeled graphs, and these models have weak compatibility with training parallelization.

In this work, we propose *Graph Generative Model with Graph Attention Mechanism* (GRAM) for generating real-world graphs that is scalable in all three contexts, especially during training. Given a set of graphs, our model approximates their distribution in an unsupervised manner. In order to achieve graph scalability, we employ an autoregressive, sequential generation process that is flexible to variable nodes and edges, formulate the likelihood of graphs in a simple manner to simplify the generation process and exploit the properties of real-world graphs such as community structure and sparseness of edges to reduce the computational cost on training. With regard to data scalability, we apply a novel graph attention mechanism that is a simple expansion of the attention mechanism used in the field of natural language processing [27] to graphs. This graph attention mechanism does not include sequentially dependent hidden states as Recurrent Neural Network (RNN) does, which improves the parallelizability of training significantly. Compared with other graph attention mechanisms such as [28, 1, 14], ours is architecturally simpler, computationally lightweight and general; it is also not limited in its applicability to generation tasks. Finally, for label scalability, we formulate the likelihood of graphs assuming multiple node/edge labels and use graph convolution and graph attention layers, the number of whose parameters do not depend directly on the number of labels.

Moreover, we introduce a non-domain-specific evaluation metric for generation tasks of node/edge-labeled graphs. Because such a method does not currently exist, prior works relied on domain-specific metrics or visual inspection, which made unified and objective evaluation difficult. Thus, we construct a general evaluation metric that dispenses with domain-specific knowledge and considers node/edge labels, combining a graph kernel and Maximum Mean Discrepancy (MMD) [12]. Although a similar statistical test based on MMD via a graph kernel is used for schema matching of protein graphs in [16], this work, to the best of our knowledge, is the first to use a graph kernel and MMD as an evaluation metric for graph generation tasks.

Our experiments on datasets of protein and molecular graphs associated with each scalability demonstrated that our models can scale up to handle large graphs and datasets that previous methods faced difficulty handling, and demonstrated results that were competitive with or superior than those of baseline methods.

In conclusion, the contributions of this work are as follows:

- We propose GRAM, a generative model for real-world graphs that is scalable especially on training.
- We propose a novel graph attention mechanism that is general and architecturally simple.
- We define scalability in graph generation tasks from three perspectives: number of nodes, data, labels.
- We construct a non-domain-specific, general evaluation metric for node/edge-labeled graph generation tasks combining a graph kernel and MMD.

2 Related Work

Although there are several traditional graph generative models [9, 3, 19, 24, 2], here, we focus on latest machine learning-based approaches, which outperform traditional methods in various graph generation tasks.

In terms of their generation process, existing graph generative models are classified into at least two types: tensor generation models and sequential generation models. Tensor generation models such as [25, 7, 13] generate a graph by outputting tensors that correspond to the graph. Although architecturally simple and easy to optimize for small graphs, these models face difficulty in generating large graphs owing to the non-unique correspondence between a graph and tensors or because of limitations in the pre-defined maximum number of nodes. In contrast, sequential generation models such as [20, 31] generate graphs by adding nodes and edges one by one, which alleviates the above problems and generates larger graphs. To achieve graph scalability, we employ the latter. However, to generate a graph with n nodes and m edges, DeepGMG [20] requires at least $\mathcal{O}(mn^2)$ operations because of its complex generation process. Although GraphRNN [31] reduces its time complexity to $\mathcal{O}(nM)$ with a constant value M, utilizing the property of breadth-first search and RNN, it cannot generate node/edge-labeled graphs mainly because it does not calculate the features of each node, relying mainly on the information in the hidden states of RNN. Moreover, these models include sequentially dependent hidden states, which makes training parallelization difficult. In contrast, our model employs a graph attention mechanism without sequentially dependent hidden states, which improves the parallelizability of training significantly. In addition, by utilizing the properties of graphs such as community structure and sparseness of edges, we reduce the complexity to almost a linear order of n while conducting a rich feature extraction.

As another classification, there are unsupervised learning approaches and reinforcement learning approaches. Given samples of graphs, unsupervised learning models [20, 31, 25, 13] approximate the distribution of these graphs in an unsupervised manner. Reinforcement learning models [23, 30, 7, 22, 21] learn to generate graphs that maximize a given objective function called return. Although reinforcement learning approaches have demonstrated promising results on several tasks such as molecular generation and network architecture optimization, we employ an unsupervised approach, which is considered to be more advantageous to the case where new samples similar to the training samples of arbitrary graphs are required or where the reward functions cannot be designed (e.g., a family of pharmaceutical molecules against a certain disease is known, but the mechanism by which they work is not entirely known).

3 Proposed Method

In this section, we first describe notations of graphs and define the graph generation task we tackle. Then, we describe the graph attention mechanism we propose, our scalable graph generative model GRAM, and its variant. GRAM approximates a distribution over graphs in an autoregressive manner and utilizes the novel graph attention mechanism to improve the parallelizability of training significantly. One step in the generation process of GRAM is illustrated in Figure 1.

3.1 Notations and Problem Definition

In this work, we define a graph $G = (V, E)$ as a data structure that consists of its node set $V = \{v_1, ..., v_n\}$ and edge set $E = \{e_{i,j} | i, j \in \{1, ..., n\}\}$. In the case of a directed graph, we define the direction of $e_{i,j}$ as from v_i to v_j. We assume nodes and edges are associated with multiple node/edge labels and denote the number of them as a and b, respectively. For graph representation, we employ tensor representation. Specifically, given node ordering π, we represent a pair comprising a graph and node ordering (G, π) as a pair of tensors (X^π, A^π). Note that π is a permutation function over $\{1, ..., n\}$. $X^\pi \in \{0, 1\}^{n \times a}$ stores information about nodes, whose i-th row is a one-hot vector corresponding to the label of $v_{\pi(i)}$. $A^\pi \in \{0, 1\}^{n \times n \times b}$ stores information about edges, whose (i, j)-th element is a one-hot vector corresponding to the label of $e_{\pi(i),\pi(j)}$. If no edge exists between $v_{\pi(i)}$ and $v_{\pi(j)}$, we replace it with zero vector. Note that (G, π) and (X^π, A^π) correspond uniquely. Finally, for simplicity, we do not consider self-looping or multiple edges, and we assume that independent elements of A^π are limited to its upper triangular elements $A^\pi_{i,j}$ $(1 \leq i \leq n, i < j \leq n)$. A possible

extension to self-looping edges is to add a step to estimate them, and for multiple edges, we can prepare additional labels that correspond to them.

The graph generation task we target is, given samples from a distribution of graphs $p(G)$, to approximate $p(G)$ in a way that we can derive new samples from it.

3.2 Graph Attention Mechanism

The aim of employing a graph attention mechanism is to efficiently take in the information of distant nodes by attention. Our graph attention mechanism is an expansion of the attention mechanism used in the field of natural language processing [27] applied to the field of graphs. One of the significant differences between graphs and sentences is that we cannot define absolute coordinates in a graph, which makes it difficult to embed positional information into nodes as in [27]. To alleviate this problem, we focus on multi-head attention in [27], and we introduce bias terms in the projection to subspaces, which are functions of the shortest path length between two nodes. Following [27], we denote matrices into which query, key, and value vectors are stacked as $Q = (\boldsymbol{q}_1, ..., \boldsymbol{q}_n)^T$, $K = (\boldsymbol{k}_1, ..., \boldsymbol{k}_n)^T$, and $V = (\boldsymbol{v}_1, ..., \boldsymbol{v}_n)^T$, respectively. In addition, we assume the i-th row corresponds to the feature vector of node v_i. With these notations, the operation in graph attention mechanism is defined as

$$\begin{aligned} \mathrm{GMultiHead}(Q, K, V) &= \mathrm{Concat}(\mathrm{head}_1, ..., \mathrm{head}_\mathrm{h}) W^O \\ \text{where } \mathrm{head}_\mathrm{i} &= \mathrm{GAttention}(Q, K, V) \end{aligned} \tag{1}$$

where h represents the number of projections to subspaces. Additionally, the operation of $\mathrm{GAttention}(\cdot, \cdot, \cdot)$ is defined as

$$\begin{aligned} \mathrm{GAttention}(Q, K, V) &= (\boldsymbol{o}_1, ..., \boldsymbol{o}_n)^T \\ \text{where } \boldsymbol{o}_i &= \textstyle\sum_j (\boldsymbol{c}_i)_j (W^V \boldsymbol{v}_j + \boldsymbol{b}^V(v_i, v_j)) \end{aligned} \tag{2}$$

Each attention weight $\boldsymbol{c}_i$ is calculated as

$$\begin{aligned} \boldsymbol{c}_i &= \mathrm{softmax}(\boldsymbol{s}_i) \\ \text{where } (\boldsymbol{s}_i)_j &= {d_k}^{-\frac{1}{2}} (W^Q \boldsymbol{q}_i + \boldsymbol{b}^Q(v_i, v_j))^T (W^K \boldsymbol{k}_j + \boldsymbol{b}^K(v_i, v_j)) \end{aligned} \tag{3}$$

The parameters to be updated are four weight matrices, $W^Q, W^K \in \mathbb{R}^{d_k \times d_{in}}$, $W^V \in \mathbb{R}^{d_v \times d_{in}}$, and $W^O \in \mathbb{R}^{d_{out} \times h d_k}$, and three bias terms, $\boldsymbol{b}^Q(\cdot, \cdot), \boldsymbol{b}^K(\cdot, \cdot) \in \mathbb{R}^{d_k}$, and $\boldsymbol{b}^V(\cdot, \cdot) \in \mathbb{R}^{d_v}$, where d_{in}, d_{out}, d_k, and d_v represent dimensions of input, output, key, and value vector, respectively. We used different parameters for calculating each head.

To consider the geometric relation between two nodes, we used functions of the shortest path length between v_i and v_j as $\boldsymbol{b}^Q(v_i, v_j)$, $\boldsymbol{b}^K(v_i, v_j)$, and $\boldsymbol{b}^V(v_i, v_j)$. Other possible approaches, for example, are using network flow instead of shortest path length or utilizing functional approximation. Furthermore, because path length is discrete, we used different weight parameters for each path length.

Specifically, setting $\boldsymbol{b}^Q = \boldsymbol{b}^K = \boldsymbol{b}^V = 0$ yields the original multi-head attention. As in [27], we added a two-layer feedforward neural network (FNN), which was applied after the above operations. We denote these operations including FNN as the graph attention mechanism in the following sections.

3.3 GRAM: Scalable Generative Models for Graphs

We first describe the likelihood formulation of graphs and then provide an overview of GRAM and its variant.

3.3.1 Likelifood Formulation of Graphs

To approximate distributions over graphs in an autoregressive manner, here, we formulate the likelihood of a pair of a graph and node ordering (G, π) and decompose it into a product of conditional probabilities. Because (G, π) uniquely defines (X^π, A^π) and vice versa, we have $p(G, \pi) = p(X^\pi, A^\pi)$, which is decomposed into a product of conditional probabilities as

$$p(X^\pi, A^\pi) = p(X^\pi_1, A^\pi_{1,1}) \prod_{s=2}^{n} p(X^\pi_s | X^\pi_{<s}, A^\pi_{<s,<s}) \prod_{t=1}^{s-1} p(A^\pi_{t,s} | A^\pi_{<t,s}, X^\pi_s, X^\pi_{<s}, A^\pi_{<s,<s}) \tag{4}$$

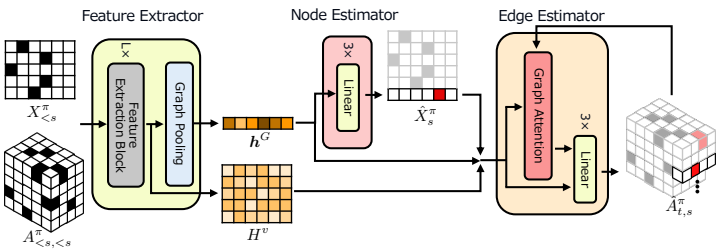

Figure 1: One step in the generation process of GRAM.

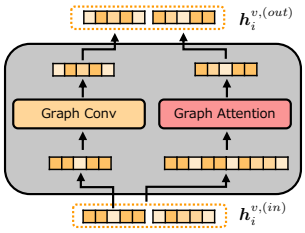

Figure 2: Feature Extraction Block.

where $A_{<s,<s}$ represents a partial tensor $A_{i,j}$ $(1 \leq i, j < s)$ and "$\cdot$" represents all indices and other notations follow as these. We omit the last dimension of each tensor for simplicity. In addition, $(X^{\pi}_{<s}, A^{\pi}_{<s,<s})$ uniquely defines a subgraph of G, which we denote as $G^{\pi}_{<s}$. To keep notations clear, we represent $(X^{\pi}_{<s}, A^{\pi}_{<s,<s})$ as $G^{\pi}_{<s}$ in the following equations. With this formulation, the likelihood of a graph G is defined as marginal, $p(G) = \sum_{\pi} p(G, \pi) = \sum_{\pi} p(X^{\pi}, A^{\pi})$.

On training, given a set of samples from $p(G)$, our GRAM approximates the joint probability $p(X^{\pi}, A^{\pi})$. Although the choice of π provides room for discussion, we adopt the breadth-first search used in [31]. More precisely, our model approximates $p(X^{\pi}, A^{\pi})$ by approximating the conditional probabilities $p(X^{\pi}_{s}|G^{\pi}_{<s})$ and $p(A^{\pi}_{t,s}|A^{\pi}_{<t,s}, X^{\pi}_{s}, G^{\pi}_{<s})$. As for $p(X^{\pi}_{1}, A^{\pi}_{1,1})$, we used a sampling distribution from training data.

On testing, we sequentially sample $\hat{X}^{\pi}_{s}$ and $\hat{A}^{\pi}_{t,s}$ from the approximated distribution and we get $(\hat{X}^{\pi}, \hat{A}^{\pi})$ when EOS is output. This can be viewed as sampling from the approximated distribution $(\hat{G}, \hat{\pi}) \sim \hat{p}(G, \pi)$. Especially by focusing only on G, we can view it as sampling from the marginal $\hat{G} \sim \hat{p}(G)$.

3.3.2 Model Overview

Here, we provide an overview of GRAM. For simplicity, we assume π to be an identity permutation (i.e., $\pi(i) = i$) in the following sections.

The architecture of GRAM consists of three networks: a feature extractor, a node estimator, and an edge estimator. The feature extractor calculates node feature vectors and a graph feature vector by summing up them, the node estimator predicts the label of the newly added node v_s, and the edge estimator predicts the labels of edges $\{e_{t,s}|t < s\}$ between previously generated nodes $\{v_t|t < s\}$ and the new node v_s. In other words, the node estimator approximates $p(X^{\pi}_{s}|G^{\pi}_{<s})$ and the edge estimator approximates $p(A^{\pi}_{t,s}|A^{\pi}_{<t,s}, X^{\pi}_{s}, G^{\pi}_{<s})$ in Equation 4.

3.3.3 Feature Extractor

Given a pair of tensors $(X^{\pi}_{<s}, A^{\pi}_{<s,<s})$, the feature extractor calculates the node feature vectors $H^{v} = (\boldsymbol{h}^{v}_{1}, ..., \boldsymbol{h}^{v}_{s-1})^{T}$ and a graph feature vector $\boldsymbol{h}^{G}$ of the corresponding subgraph $G^{\pi}_{<s}$. It consists of L feature extraction blocks and a subsequent graph pooling layer. We used $L = 3$ in the experiments.

A feature extraction block is composed of a graph convolution layer and a graph attention layer stacked in parallel, as illustrated in Figure 2. To keep compatibility with the community-oriented variant described in Section 3.4, we used divided feature vectors to keep the output of graph attention layers from flowing into the input of the graph convolution layers. We aim to extract local information by using graph convolution layers and global information by using graph attention layers. Although there are various types of graph convolutions, we employed the one used in [15]. [1] Roughly, the selected graph convolution type convolutes the features of neighboring nodes and edges into each node and edge in a graph. A graph attention layer operates self-attention, in which query/key/value vectors are all node feature vectors. To reduce the computational cost, we restricted the range of attention to r-neighboring nodes. Also, to exploit low-level features of graphs, we stacked degree,

[1] In [15], directed graphs are assumed as input. In the case of undirected graphs, we consider only g_s and ignore g_o, illustrated in Figure 3 in [15].

clustering coefficient, and distance from the center of the graph into each node vector, which are all non-domain-specific statistics in graphs.

The graph pooling layer computes a graph feature vector $\boldsymbol{h}^G$ by summing up all node feature vectors in the graph. To improve its expressive power in aggregation, we used a gating network as in [20]. Specifically, the operation in the graph pooling layer is defined as

$$\boldsymbol{h}^G = \sum_{i=1}^{s-1} \sigma(g_{pool}(\boldsymbol{h}_i^v))\boldsymbol{h}_i^v \tag{5}$$

where g_{pool} is a two-layer FNN and σ is a sigmoid function.

3.3.4 Node Estimator

The node estimator predicts the label of the new node v_s. Concretely, it computes $\hat{X}_s^\pi$ from $\boldsymbol{h}^G$ as

$$\hat{X}_s^\pi = g_{NE}(\boldsymbol{h}^G) \tag{6}$$

where g_{NE} is a three-layer FNN, the dimension of whose last layer is $a+1$ including EOS and the activation function is a softmax function. We terminate the generation process when EOS is output.

3.3.5 Edge Estimator

The edge estimator predicts the labels of the edges $\{e_{t,s}|t<s\}$ between previously generated nodes $\{v_t|t<s\}$ and the new node v_s. More precisely, it computes $\hat{A}_{t,s}^\pi$ from $\boldsymbol{h}_s^v, \boldsymbol{h}_t^v, \boldsymbol{h}^G$ and previously predicted edge labels as

$$\hat{A}_{t,s}^\pi = g_{EE}(\boldsymbol{h}_s^v, \boldsymbol{h}_{<t}^e, \boldsymbol{h}_t^v, \boldsymbol{h}^G) \quad (t=1,...,s-1) \tag{7}$$

where $\boldsymbol{h}_s^v$ is the embedded vector of the predicted label of v_s. $\boldsymbol{h}_{<t}^e$ is calculated by a graph attention in which the query vector is $\mathrm{Concat}(\boldsymbol{h}_s^v, \boldsymbol{h}_t^v, \boldsymbol{h}^G)$ and the value/key vectors are $\{\mathrm{Concat}(\boldsymbol{h}_s^v, \boldsymbol{h}_{j,s}^e, \boldsymbol{h}_j^v, \boldsymbol{h}^G)|j<t\}$, where $\boldsymbol{h}_{j,s}^e$ denotes the embedded vector of the predicted label of $e_{j,s}$. Thereby, we aim to express the dependency to $A_{<t,s}^\pi$ in $p(A_{t,s}^\pi|A_{<t,s}^\pi, X_s^\pi, G_{<s}^\pi)$. We use a three-layer FNN as g_{EE}, in which the dimension of the last layer is $b+1$ including EOS with a softmax function as the activation function. When EOS is output, we do not add an edge.

However, to generate a graph with n nodes, the edge estimation process requires $\mathcal{O}(n)$ operations for one edge, resulting in $\mathcal{O}(n^2)$ operations for one step and $\mathcal{O}(n^3)$ operations in total, which is a significant obstacle to achieving graph scalability. Therefore, we resort to a remedy obtained from empirical observation. Specifically, our inspection on each attention weight in the edge estimation showed that most of the weights for nodes that were predicted to have no edge between v_s (i.e., $\{v_j|\hat{A}_{j,s}^\pi = \mathbf{0}, j<t\}$) were nearly zero, while the weights for nodes that were predicted to have edges (i.e., $\{v_j|\hat{A}_{j,s}^\pi \neq \mathbf{0}, j<t\}$) took larger values. This observation suggests that among $\{v_j|j<t\}$, those that have edges between v_s are important and the others are not in the prediction of $A_{t,s}^\pi$. Hence, we deterministically set these weights to zero, which means we do not consider them in graph attention. With this approximation (i.e., $p(A_{t,s}^\pi|\{A_{j,s}^\pi|A_{j,s}^\pi \neq \mathbf{0}, j<t\}, X_s^\pi, G_{<s}^\pi) \sim p(A_{t,s}^\pi|A_{<t,s}^\pi, X_s^\pi, G_{<s}^\pi)$), we reduce the number of required operations to $\mathcal{O}(n^2\alpha)$, where $\alpha = |\{v_j|\hat{A}_{j,s}^\pi \neq \mathbf{0}, j<t\}|$. In addition, because $\alpha \leq \deg(v_s)$ holds, we can assume $\alpha \ll n$ when $\deg(v_s) \ll n$, which is often the case with many real-world graphs.

3.4 cGRAM: Community-Oriented Variant

In addition to GRAM described above, here we present a community-oriented variant, cGRAM, in which we aimed to reduce the computational cost significantly by posing restrictions about community structures.

In real-world graphs, the distribution of edges is biased, not uniform, which allows us to divide the node set into subsets called community, in which nodes are densely connected with each other [11]. We utilize this property to reduce computational complexity. Specifically, denoting communities in a subgraph $G_{<s}^\pi$ as $\{C_1,...,C_l\}$, in this model, we assume that the nodes that have edges between the new node v_s are restricted to within the same community C_j and that the label of v_s depends only on C_j. This assumption is mainly based on the property of community structure, where intra-community

connections are more dense than inter-community connections and that nodes in the same community have stronger relations than those in other communities [11]. Concretely, we modify the graph pooling of Equation 5 to a community pooling defined as

$$\boldsymbol{h}_j^C = \sum_{v_i \in C_j} \sigma(g_{pool}(\boldsymbol{h}_i^v))\boldsymbol{h}_i^v \tag{8}$$

and we replace $\boldsymbol{h}^G$ with $\boldsymbol{h}_j^C$ in the subsequent operations. Additionally, we restrict the range of graph attention to the same community. With these assumptions and restrictions, on training we only have to consider nodes in C_j and their L-neighbors in the feature extraction, the node estimation, and the edge estimation, which improves computational performance significantly. On testing, we assume the probability that v_s belongs to each community is equal (i.e., $p(v_s \in C_j) = 1/l$ $(j = 1, ..., l)$) and select the target community randomly.

4 Experiments

To evaluate the performance and scalability of our models, we compared GRAM and cGRAM with prior graph generative models in the generation tasks for two types of real-world graphs, protein graphs and molecular graphs, which correspond to graph scalability and data scalability, respectively. As for label scalability, in this work, we only considered whether the model could handle multiple node/edge labels or not because we thought the performance would be strongly affected by its architecture design and thus fair comparison is difficult.

4.1 Protein Graphs: Graph Scalability

To evaluate graph scalability, we performed an experiment using protein graphs that contain a relatively large number of nodes. Basically, the experiment settings followed [31].

We used a dataset of 918 protein graphs [8], in which each node corresponds to an amino acid and two nodes that are chemically connected or spatially close are connected by an edge. In detail, the number of nodes is $100 \leq n \leq 500$ and the number of node/edge labels is $a = 89, b = 1$. We allocated 80% of the graphs for training and the rest for testing.

We compared our models with GraphRNN, GraphRNN-S [31] as recent deep learning-based baselines. In addition, we reported the results of Erdös-Rényi model (E-R) [9], Barabási-Albert (B-A) [3], Kronecker Graphs [19], and Mixed Membership Stochastic Blockmodels (MMSB) [2] as traditional baselines.

To measure the quality of the generated graphs, we used MMD [12] for three types of graph features: degree, clustering coefficient, and orbit counts, proposed in [31]. These metrics allowed us to measure the distance between the distribution of the generated graphs and that of real graphs quantitively. We reported the average of the squared MMD score with three runs.

Table 1 summarizes the results.[2] Our models performed better than most of the baselines in terms of clustering coefficient and orbit counts, and cGRAM was the best. Considering that these are middle/high-level features of graphs, this is likely due to employing a graph attention in the edge estimation, which can consider the relative position between two nodes. However, our models performed poorly in terms of degree compared with GraphRNN, which is likely because the way that nodes are connected involves their spatial proximity; because this is difficult to estimate and noisy, it is possible that the rich feature extraction backfired and thus it failed to capture low-level features. On average, cGRAM performed the best. We presume the reason is that focusing on one community made training stable and led to better performance because the size of a community is smaller and relatively more constant than that of a graph.

4.2 Molecular Graphs: Data Scalability

To evaluate data scalability, we conducted an experiment using a dataset that consists of 250k molecular graphs, following [20].

[2] We used the code provided by `http://github.com/snap-standard/GraphRNN`. To correctly calculate the MMD score and for equal comparison, we omit the cleaning post-process in the code and re-evaluated all models.

Table 1: Comparison of the squared MMD score of degree, clustering coefficient, and orbit counts.

	Deg.	Clus.	Orbit
E-R [9]	0.154	1.788	1.098
B-A [3]	1.452	1.713	0.914
Kronecker [19]	1.048	1.799	0.668
MMSB [2]	0.623	1.793	1.261
GraphRNN-S [31]	0.274	0.293	0.204
GraphRNN [31]	**0.087**	1.055	0.774
GRAM (ours)	0.257	0.431	0.101
cGRAM (ours)	0.171	**0.119**	**0.077**

Table 2: Comparison of the percentage of valid and novel samples and the squared GK-MMD score.

	Valid	Novel	GK-MMD
DeepGMG [20]	89.2	89.1	0.0200
GRAM (ours)	94.0	94.0	**0.0192**
cGRAM (ours)	**96.1**	**94.4**	0.0214

We used the 250k samples from ZINC dataset [26] provided by [17], where nodes and edges correspond to heavy atoms and chemical bonds, respectively. The number of nodes is $6 \leq n \leq 38$ and the number of node/edge labels is $a = 9, b = 3$.

We used DeepGMG [20] as a baseline.[3] With an emphasis on the ability to approximate distributions of arbitrary real-world graphs, we excluded other graph generative models that use domain-specific knowledge or are based on reinforcement learning.

Following [20], we generated 100k graphs and calculated the percentages of graphs that were valid as molecules and the percentage of novel samples that did not appear in the training set. In addition, combining a graph kernel and MMD, we constructed a general evaluation metric to measure the distance between the distribution of the generated graphs and that of real graphs, which are both node/edge labeled.

MMD is a test statistic to determine whether two sets of samples from distribution p and q are derived from the same distribution (i.e., $p = q$), and especially when its function class $\mathcal{F}$ is a unit ball in a reproducing kernel Hilbert space (RKHS) $\mathcal{H}$, we can derive the squared MMD as

$$\mathrm{MMD}^2[\mathcal{F}, p, q] = \mathbb{E}_{x,x' \sim p}[k(x, x')] - 2\mathbb{E}_{x \sim p, y \sim q}[k(x, y)] + \mathbb{E}_{y,y' \sim q}[k(y, y')] \tag{9}$$

where $k(\cdot, \cdot)$ is the associated kernel function [12].

Graph kernels are kernel functions over graphs, and we used Neighborhood Subgraph Pairwise Distance Kernel (NSPDK) [6], which measures the similarity of two graphs by matching pairs of subgraphs with different radii and distances. Because NSPDK is a positive-definite kernel [6], it follows that it defines a unique RKHS $\mathcal{H}$ [4]. This fact allows us to calculate the squared MMD using Equation 9. Moreover, because NSPDK considers node/edge labels, we used this graph kernel MMD (GK-MMD) as an evaluation metric for node/edge-labeled graph generation tasks. To reduce the computational cost, we sampled 100 graphs from the generated and the real graphs and then reported the average squared GK-MMD with 10 runs.

Table 2 lists the results. GRAM and cGRAM achieved higher valid and novel percentages, and GRAM performed the best in terms of GK-MMD, which is likely due to the rich feature extraction and the consideration of the distance between nodes when estimating edges. We suppose the poor performance of cGRAM on GK-MMD is due to the restrictions on community structure because molecular graphs are relatively small.

5 Conclusion

In this work, we tackled the problem of scalability as it is one of the most important challenges in graph generation tasks. We first defined scalability from three perspectives, and then proposed a scalable graph generative model, GRAM, and its variant. In addition, we proposed a novel graph attention mechanism as a key portion of the model and constructed graph kernel MMD as a general evaluation metric for node/edge-labeled graph generation tasks. In our experiment, which used protein graphs and molecular graphs, we verified the scalability and competitive or superior performances of our models.

[3] To evaluate DeepGMG in GK-MMD, we used generated molecular graphs provided by the author of [20].

Acknowledgments

This work was supported by JST CREST Grant Number JPMJCR1403, Japan. We would like to thank Atsuhiro Noguchi for his helpful advice on graphs. We would also like to show our gratitude to Yujia Li for providing data of generated molecular graphs and giving permission to use it for comparison in the experiment. Finally, we would like to appreciate every member in the lab for beneficial discussions, which inspired our work greatly.

References

[1] Abu-El-Haija, S., Perozzi, B., Al-Rfou, R., and Alemi, A. A. (2018). Watch your step: Learning node embeddings via graph attention. In *Advances in Neural Information Processing Systems 31*.

[2] Airoldi, E. M., Blei, D. M., Fienberg, S. E., and Xing, E. P. (2009). Mixed membership stochastic blockmodels. In *Advances in Neural Information Processing Systems 21*.

[3] Albert, R. and Barabási, A.-L. (2002). Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74:47–97.

[4] Aronszajn, N. (1950). Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68(3):337–404.

[5] Blondel, V. D., Guillaume, J.-L., Lambiotte, R., and Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008.

[6] Costa, F. and Grave, K. D. (2010). Fast neighborhood subgraph pairwise distance kernel. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*.

[7] De Cao, N. and Kipf, T. (2018). MolGAN: An implicit generative model for small molecular graphs. *arXiv e-prints*, page arXiv:1805.11973.

[8] Dobson, P. D. and Doig, A. J. (2003). Distinguishing enzyme structures from non-enzymes without alignments. *Journal of Molecular Biology*, 330(4):771 – 783.

[9] Erdös, P. and Rényi, A. (1959). On random graphs, i. *Publicationes Mathematicae (Debrecen)*, 6:290–297.

[10] Fortunato, S. (2010). Community detection in graphs. *Physics reports*, 486(3-5):75–174.

[11] Fortunato, S. and Castellano, C. (2012). Community structure in graphs. *Computational Complexity: Theory, Techniques, and Applications*, pages 490–512.

[12] Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., and Smola, A. (2012). A kernel two-sample test. *J. Mach. Learn. Res.*, 13:723–773.

[13] Grover, A., Zweig, A., and Ermon, S. (2018). Graphite: Iterative Generative Modeling of Graphs. *arXiv e-prints*, page arXiv:1803.10459.

[14] Ishiguro, K., Maeda, S.-i., and Koyama, M. (2019). Graph Warp Module: an Auxiliary Module for Boosting the Power of Graph Neural Networks. *arXiv e-prints*, page arXiv:1902.01020.

[15] Johnson, J., Gupta, A., and Fei-Fei, L. (2018). Image generation from scene graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[16] Kriegel, H.-P., Borgwardt, K. M., Gretton, A., Schölkopf, B., Rasch, M. J., and Smola, A. J. (2006). Integrating structured biological data by Kernel Maximum Mean Discrepancy. *Bioinformatics*, 22(14):e49–e57.

[17] Kusner, M. J., Paige, B., and Hernández-Lobato, J. M. (2017). Grammar Variational Autoencoder. *arXiv e-prints*, page arXiv:1703.01925.

[18] Lancichinetti, A. and Fortunato, S. (2009). Community detection algorithms: a comparative analysis. *Physical review E*, 80(5):056117.

[19] Leskovec, J., Chakrabarti, D., Kleinberg, J., Faloutsos, C., and Ghahramani, Z. (2010). Kronecker graphs: An approach to modeling networks. *J. Mach. Learn. Res.*, 11:985–1042.

[20] Li, Y., Vinyals, O., Dyer, C., Pascanu, R., and Battaglia, P. (2018). Learning Deep Generative Models of Graphs. *arXiv e-prints*, page arXiv:1803.03324.

[21] Li, Y., Zhang, L., and Liu, Z. (2018). Multi-objective de novo drug design with conditional graph generative model. *Journal of Cheminformatics*, 10(1):33.

[22] Liu, Q., Allamanis, M., Brockschmidt, M., and Gaunt, A. (2018). Constrained graph variational autoencoders for molecule design. In *Advances in Neural Information Processing Systems 31*.

[23] Luo, R., Tian, F., Qin, T., Chen, E., and Liu, T.-Y. (2018). Neural architecture optimization. In *Advances in Neural Information Processing Systems 31*.

[24] Robins, G., Pattison, P., Kalish, Y., and Lusher, D. (2007). An introduction to exponential random graph (p*) models for social networks. *Social Networks*, 29(2):173 – 191.

[25] Simonovsky, M. and Komodakis, N. (2018). Graphvae: Towards generation of small graphs using variational autoencoders. In *ICANN*.

[26] Sterling, T. and Irwin, J. J. (2015). Zinc 15 – ligand discovery for everyone. *Journal of Chemical Information and Modeling*, 55(11):2324–2337.

[27] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems 30*.

[28] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018). Graph Attention Networks. *International Conference on Learning Representations*.

[29] Wang, H., Wang, J., Wang, J., Zhao, M., Zhang, W., Zhang, F., Xie, X., and Guo, M. (2017). GraphGAN: Graph Representation Learning with Generative Adversarial Nets. *arXiv e-prints*, page arXiv:1711.08267.

[30] You, J., Liu, B., Ying, Z., Pande, V., and Leskovec, J. (2018a). Graph convolutional policy network for goal-directed molecular graph generation. In *Advances in Neural Information Processing Systems 31*.

[31] You, J., Ying, R., Ren, X., Hamilton, W. L., and Leskovec, J. (2018b). Graphrnn: Generating realistic graphs with deep auto-regressive models. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*.

A Appendix

A.1 Complexity Analysis

In this section, we compare the computational complexity on training of our models with that of existing models. Following [27], we consider two demands: the total computational complexity and the minimum sequential operations with parallelization. Note that we assume that all matrix-vector products can be conducted with $\mathcal{O}(1)$ time complexity. Table 3 summarizes the evaluation results.

To generate a graph with n nodes and m edges, DeepGMG [20] requires at least $\mathcal{O}(mn^2)$ operations. Because the initialization of the state of the new node depends on the state in the previous step, the amount of its minimum sequential operations is $\mathcal{O}(n^2)$. On the other hand, GraphRNN [31] requires $\mathcal{O}(nM)$ operations with a constant value M, utilizing the properties of breadth-first search and relying mainly on the hidden states of RNN. However, the number of its minimum sequential operations is $\mathcal{O}(n + M)$ because of the sequential dependency of the hidden state of RNN; moreover, it cannot generate node/edge-labeled graphs.

Next, we evaluate GRAM. We denote the average number of r-neighboring nodes as N_r and reuse α, defined in Section 3.3.5. Focusing on one step of the generation process, The feature extractor requires $\mathcal{O}(m + nN_r)$ operations, considering a graph convolution layer requires $\mathcal{O}(m)$, and a graph attention layers requires $\mathcal{O}(nN_r)$. In addition, the node estimator requires $\mathcal{O}(1)$ operations, and the edge estimator requires $\mathcal{O}(n\alpha)$ operations. Therefore, the total complexity of GRAM is $\mathcal{O}(n(m + n(N_r + \alpha)))$. Note that all the estimation operations of the node estimator and the edge estimator can be parallelized on training because our model has no sequentially dependent hidden states as does RNN. As for cGRAM, we can similarly derive its total complexity as $\mathcal{O}(n(m_c + n_c(N_r + \alpha)))$, where n_c and m_c denote the average number of nodes and edges, in one community, respectively.

From the above analysis, we can say that our GRAM, while keeping rich feature extraction, requires fewer operations than DeepGMG when $N_r, \alpha \ll n$ holds, which is the often case in many real-world graphs. In addition, assuming the size of each community is nearly constant and smaller than that of a graph, the total complexity of cGRAM can be regarded as almost linear of n, which is competitive to GraphRNN, while ours keeps feature extraction. More importantly, all the estimation operations can be parallelized, which facilitates training with multiple computing nodes. Therefore, we can expect graph scalability and data scalability of our models. We also expect label scalability because our models are flexible to variable number of labels by modifying only the dimensions of the input and output layers.

Table 3: Total computational complexity and minimum sequential operations on training.

Method	Total Complexity	Minimum Sequential Operations
DeepGMG [20]	$\mathcal{O}(mn^2)$	$\mathcal{O}(n^2)$
GraphRNN [31]	$\mathcal{O}(nM)$	$\mathcal{O}(n + M)$
GRAM (ours)	$\mathcal{O}(n(m + n(N_r + \alpha)))$	$\mathcal{O}(1)$
cGRAM (ours)	$\mathcal{O}(n(m_c + n_c(N_r + \alpha)))$	$\mathcal{O}(1)$

A.2 Inspection on Attention Weights in Edge Estimation

In a brief experiment on protein graphs where we consider all of the previously generated nodes in edge estimation (i.e. its total computational complexity is $\mathcal{O}(n^3)$), we examined the distributions of attention weights for nodes that are predicted to have edges between the new node and for those not. Figure 3 shows the two distributions of attention weights in one step in edge estimation (i.e., in the calculation of $\boldsymbol{h}^e_{<t}$). The blue one represents the distribution of weights for nodes that are predicted to not have edges between the new node v_s (i.e. $\{v_j | A^\pi_{j,s} = \mathbf{0}, j < t\}$), and the orange one represents the distribution of weights for nodes that are predicted to have edges between the new node (i.e. $\{v_j | A^\pi_{j,s} \neq \mathbf{0}, j < t\}$). From the figure, we can see that the former distribution is sharper around the zero compared with the latter, most of whose weights take nearly zero. As for molecular graphs, we observed a similar result.

A.3 Comunity Detection Algorithm

Here we describe the community detection algorithm used in this work. Although it is possible to include community detection into the learning framework, we did not consider this option for simplicity.

While various algorithms to detect communities in a graph are proposed [18], we use the algorithm modified from Louvain algorithm [5], which is simple and computationally lightweight. Louvain algorithm detects communities by maximizing modularity $L_Q \in [-1, 1]$ in a greedy manner. Modularity L_Q is a function to

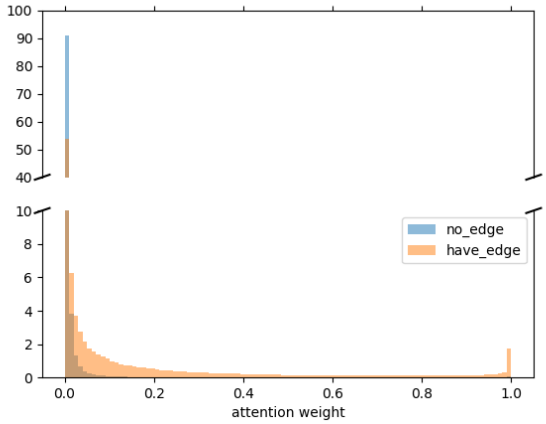

Figure 3: The distributions of attention weights. The blue one represents that of previously generated nodes that are predicted to not have edges with the new node, and the orange one represents that of nodes that are predicted to have edges. Note that the scale of y-axis is different between the top half and the bottom half.

measure the quality of partition and is calculated as:

$$L_Q = \frac{1}{2m} \sum_{i,j} \left[A'_{i,j} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j) \tag{10}$$

where $A'_{i,j}$ denotes the weight of the edge between v_i and v_j, c_i represents the community that v_i belongs to, and $k_i = \sum_j A'_{i,j}, m = \frac{1}{2} \sum_{i,j} A'_{i,j}$ [5]. Note that $\delta(c_i, c_j) = 1$ if $c_i = c_j$ and $\delta(c_i, c_j) = 0$ otherwise. Specifically, it leverages the fact that the change in modularity ΔL_Q by adding/deleting a node to/from a community can be calculated easily.

However, we found using this algorithm simply can produce quite small communities, which makes training unstable. To alleviate this problem, we introduced coverage $L_C \in [0, 1]$ [10] as a reguralization term. Coverage L_C is defined as a ratio of inner-community edges to all edges. More precisely, we modify the objective function of Louvain algorithm to $L_Q + \lambda L_C$ with a regularization parameter λ. Note that setting $\lambda = 0$ yields the original Louvain algorithm and setting $\lambda \to \infty$ returns only one community without partitioning. Even with this modification, the change $\Delta(L_Q + \lambda L_C)$ can also be calculated quite simply, hence its time complexity is almost the same as the original. We set $\lambda = 2.0$ for molecular graphs and $\lambda = 4.0$ for protein graphs.

A.4 Visual Inspection on Generated Molecular Graphs

For qualitative evaluation, we listed molecules in the training set and those generated by each model in Figure 4, 5, 6, 7. As for GRAM and DeepGMG, we could not see the significant difference from training set, whereas cGRAM generated some small molecules, which was considered to be captured in the poor performance in GK-MMD.

A.5 Detail of Experiment Settings

For convenience of implementation, we started the generation process from a small subgraph with n_{min} nodes (i.e., replace $(X^\pi_1, A^\pi_{1,1})$ with $(X^\pi_{n_{min}}, A^\pi_{n_{min},n_{min}})$ in Equation 4). We set $n_{min} = 12$ for protein graphs and $n_{min} = 6$ for molecular graphs.

In the experiment of protein graphs, we used Tesla V100 $\times$ 4 for training. We trained the models with batch size 2048 through 200 epoch. It took about 10 hours for GRAM and 5 hours for cGRAM. We set $r = 4$.

In the experiment of molecular graphs, we used Tesla V100 $\times$ 2 for training. We trained the models with batch size 8192 through 40 epoch. It took about 10 hours for GRAM and 8 hours for cGRAM. We set $r = 7$. Although 250k samples are limited amount compared with a large number of molecular graphs, this training time indicates the scalability to larger datasets.

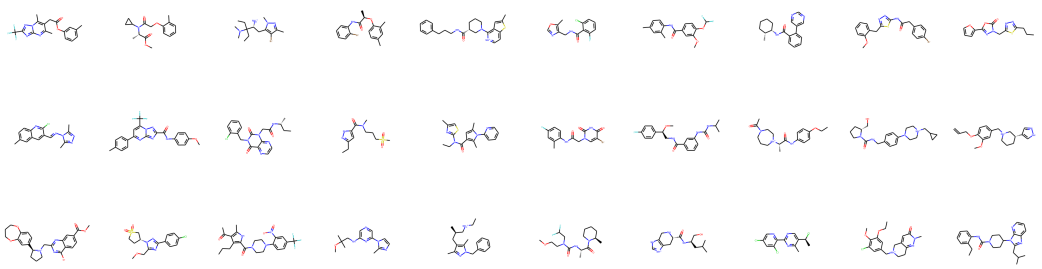

Figure 4: 30 samples from training set.

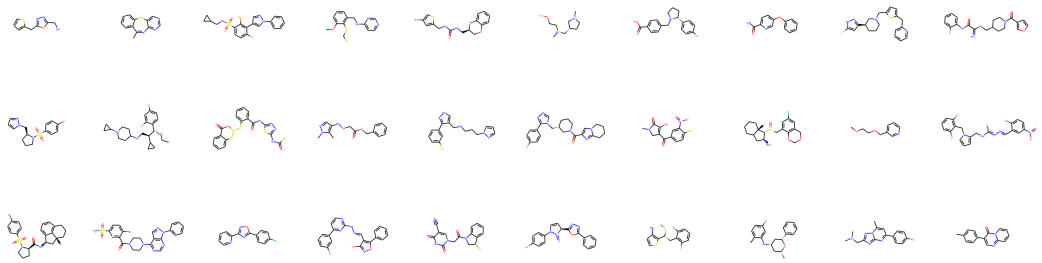

Figure 5: 30 samples from molecules generated by DeepGMG.

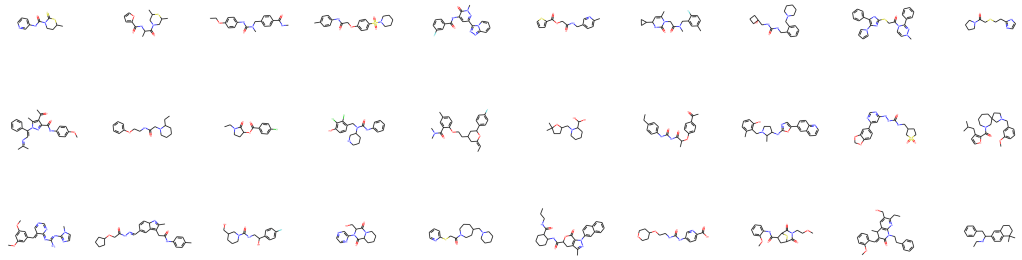

Figure 6: 30 samples from molecules generated by GRAM.

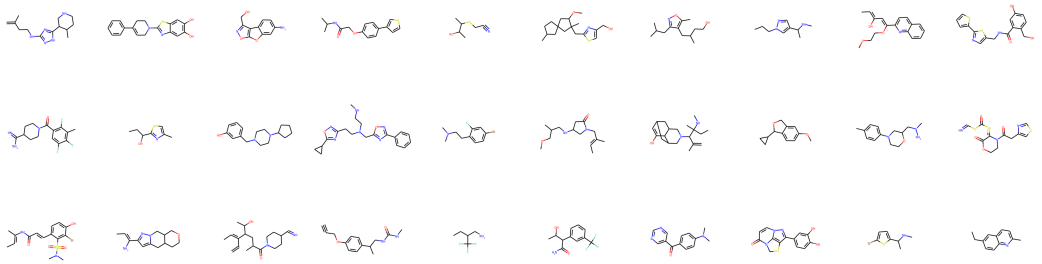

Figure 7: 30 samples from molecules generated by cGRAM.