Subgraph-based Machine Learning for Graph Generation

Masatsugu Yamada

Doctor of Philosophy

Department of Informatics
School of Multidisciplinary Sciences

The Graduate University for Advanced Studies, SOKENDAI

September 2023

A dissertation submitted to Department of Informatics
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Advisory Committee

Assoc. Prof. Mahito SUGIYAMA	National Institute of Informatics, The Graduate University for Advanced Studies, SOKENDAI
Prof. Takeaki UNO	National Institute of Informatics, The Graduate University for Advanced Studies, SOKENDAI
Prof. Katsumi INOUE	National Institute of Informatics, The Graduate University for Advanced Studies, SOKENDAI
Prof. Ken SATOH	National Institute of Informatics, The Graduate University for Advanced Studies, SOKENDAI
Assoc. Prof. Hiroko SATOH	University of Zurich, Research Organization of Information and Systems, ROIS

Acknowledgements

I want to express my gratitude to many individuals, institutions, and the company who contributed in many different ways to support me.

First and foremost, I want to express my sincere gratitude to my supervisor, Assoc. Prof. SUGIYAMA Mahito for continuous support and invaluable advice on my research. It would have been impossible to finish my dissertation without his persistent help. He gave lectures on graph kernel, graph mining and mindsets towards research. I am greatly thankful for all that and his supports in the course.

Besides, I wish to acknowledge the rest of my Ph.D. committee: Prof. UNO Takeaki, Prof. INOUE Katsumi, Prof. SATOH Ken, and Assoc. Prof. SATOH Hiroko and RIKEN AIP Dr. TAKIGAWA Ichigaku for their insightful and helpful comments. In addition, my special thanks are due to secretary TOKUDA Hiroko for helping me with many things, such as procurement of supplies, accounting procedures, etc.

I am grateful to my ex-managers and ex-colleagues at AGC Inc., Innovative Technology Laboratory. I want to express my sincere gratitude to AGC for supporting me when I worked at the time. In particular, a team leader TODORIKI Hiroshi, managers, KITAYAMA Daisuke, and MINAMI Takuya allowed me to take a Ph.D. and allocate extra time for studying while working in the company.

I would like to thank all the members in Sugiyama laboratory, ABE Hiroto, MUHAMMAD Masykur, MIZUGUCHI Makoto, MATSUE Kiyotaka, LUO Simon Junming, CHEEMA Prasad, PÂRT, ACHI Profir-Petru, AHMED Md. Sohel, GHALAMKARI Kazu, KANOH Ryuichi, KAWAKAMI Yuhi, ISHIZAKI Ryunosuke, MARUYAMA Yuichi. They gave me a lot of invaluable advice in the Lab seminar and Lunch seminar.

Last but not least, I am grateful to my family for supporting me through my life.

Masatsugu Yamada
September 2023

Abstract

Graph-structured data are ubiquitous in the real world, ranging from social networks, supply chains, and transportation networks to bio and chemo-informatics. There are various graph machine learning tasks to analyze graph datasets. Some examples of graph machine learning tasks include link prediction, node classification, graph embedding, graph regression, and graph classification. More and more algorithms have been proposed in order to achieve these tasks. Especially in the case of chemo-informatics domain, molecules can be represented as graphs, and various methods to embed molecular graphs into latent space have been proposed to analyze molecular properties and similarity between molecules. It is essential to learn the representation of graph structures and generate novel molecules with desired properties demanded in drug discovery and material science. However, generating graphs with desired properties is highly challenging due to the combinatorial optimization problem, satisfying the strict condition of the definition of graphs. As for the case of image generation, deep generative models such as Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs) are often employed, and the generated image does not need to be reconstructed pixel by pixel rigorously. In graph generation tasks, however, although generative models are also often used, if the generated graph has a different bit in the adjacency matrix, the graph would be completely different from the original graph to reconstruct. This is because, especially in a molecular graph, it is not likely to satisfy the definition of molecules because the node (atom) must have a limited number of edges (degree) according to the atom types. For example, if an atom is a carbon, the number of degrees in the node of interest must be up to four, and the generated graph from a deep generative model often violates the rule.

In the first chapter of this thesis, we state the outlines of the preliminary graph theory and the peripheral areas to understand graph generation, particularly molecular graphs. Graph theory is indispensable for representing graphs appropriately. Graph generation covers a broad range of machine learning topics: deep generative models for generating data, graph kernels and graph neural networks for predicting target properties, and reinforcement learning for generating graphs to guide with target properties. We also state the representation of molecules as strings and

graphs. String notation of molecules helps treat the chemical structures and is also helpful for applying natural language processing techniques.

In the second chapter of this thesis, we state the molecular graph generation algorithm based on subgraphs. We propose a novel method called the MOLDR (MOLecular graph Decomposition and Reassembling) algorithm to generate molecular graphs by combining subgraphs mined from molecular graph datasets by using the subgraph mining algorithm and searching molecules with target properties via Monte Carlo tree search and reinforcement learning. Our method can generate molecules maximized with respect to logP and QED, which are used as benchmarks for molecular graph generation. In contrast to deep generative models, MOLDR can generate molecular graphs directly so that the generating process is interpretable when evaluating the path.

In the third chapter of this thesis, we investigate how the subgraph features from message passing affect graph classification and regression. This is also related to generating graphs with desired properties. This is because when generating graphs, it is necessary to use some objective function to maximize or minimize target properties, which is basically a surrogate model, that is, the property prediction function, trained on a graph dataset. Most proposed graph neural networks are based on a message passing scheme, aggregating the features in neighboring nodes. This operation is similar to the Weisfeiler–Lehman graph isomorphism test for discerning whether or not two graphs are isomorphism. The algorithm is related to focusing on the subgraph structure through a message passing scheme. Our main contributions are that the WL kernel outperforms the performance of classification and regression over the most fundamental graph neural networks inspired by message passing schemes, such as graph convolutional networks (GCNs) and graph isomorphism networks (GINs). We also investigate the effect of a bigger size of subgraph structures by iterating the number of message passing schemes. The performance of the WL kernel does not deteriorate even if the message passing iteration increases. In contrast, the performance of GCNs and GINs deteriorates due to the large number of parameters to train and ill-trained previous features.

In the final chapter, we summarize our contributions to this dissertation and discuss future work of molecular graph generation and the problem between molecular graph generation and the objective function of GNNs with the larger number of message passing iterations. Although MOLDR can generate molecules with maximized target properties, we need to investigate further the strengths and weaknesses of MOLDR on other benchmark datasets. MCTS needs more rollout to search molecules, so we need to speed up searching and apply another reinforcement learning with parallel computations to MOLDR.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Designing de novo molecules for drugs and materials with desired properties is a highly challenging task due to the combinatorial problem of finding desired graphs. Molecules can be essentially represented as *graphs* with node and edge attributes. Hence, graph mining methods and graph machine learning have been applied for this task and enumerating substructures in molecules. In order to discover drug candidates, one of the most fundamental approaches is using the quantitative structure-activity relationship (QSAR), a mathematical model to explain relationships between biological activities and the structural properties of molecules. A QSAR model is often used to do high-throughput screening of molecules. It is helpful to search within the dataset researchers possess and the molecules they come up with. However, to design new molecules, there needs to solve an inverse problem of QSAR models, and it is challenging to solve the optimization problem because a QSAR model is trained with molecular descriptors computed from graph structures, such as the molecular weight (sum of the node features in a graph (each atomic weight in a molecule)). After converting graphs into descriptors, the information on graph structure is lost. Hence, the optimization through the QSAR model results in optimized descriptors and cannot generate graphs directly. Several methods have been proposed to tackle this problem of molecular graph generation. Recent advanced approaches to finding drug-candidate molecules have employed deep generative models. The basic idea of using generative models is to learn the latent representation of molecules, which enables latent vectors to be reconstructed and explore molecules that satisfy target properties in the learned latent chemical space. Exploration methods such as Bayesian optimization are used to search the latent chemical space. However, it is fundamentally difficult to reconstruct molecular graphs from the latent space and search for molecules with the desired

property by extrapolation from a training dataset, as a large part of the latent space represents invalid molecules. There is another problem when using graph neural networks (GNNs) for embedding graphs and surrogate models for predicting target properties. Many proposed GNN models are based on message passing algorithms, which is the fundamental approach to extracting features of subgraph structures by aggregating information in neighboring nodes. This operation is related to the Weisfeiler-Lehman graph isomorphism test for discerning whether two graphs are isomorphisms. When the number of message-passing iterations increases, the problem has been reported that GNNs fail to represent node features in graphs. It is a critical problem when incorporating GNNs into graph generation algorithms for learning the representation of graph structures and estimating the reward function that maximizes a target in reinforcement learning. This first chapter states the fundamentals of graphs and machine learning for molecular graphs and the methods to generate and search methods to obtain molecular graphs with desired properties.

1.2 Graph theory

First, we define terminology and notation of graphs necessary for explaining graph generation and graph machine learning.

A *graph* is a tuple $G = (V, E)$, where V and E denote the set of nodes $V = \{v_1, v_2, \ldots, v_n\}$ and edges $E = \{e_1, e_2, \ldots, e_m\}$, respectively. Edges are a tuple of neighboring nodes (v_i, v_j). The number of vertices and edges of G, the cardinality of V and E, are represented as $|V|$ and $|E|$, respectively. Nodes and edges can have labels (attributes) via label functions $l_V : V \to \Sigma_V$ for nodes and $l_E : E \to \Sigma_E$ for edges with some label domain Σ_V and Σ_E, which can be any set such as $\mathbb{Z}$ and $\mathbb{R}^d$. A labeled graph is a triple $G = (V, E, \mathcal{L})$, where $\mathcal{L}$ is the set of node label and edge label obtained from l_V and l_E respectively. For two graphs $G = (V, E)$ and $G' = (V', E')$, we say that G' is a *subgraph* of G, denoted by $G' \sqsubseteq G$, if $V' \subseteq V$ and $E' \subseteq (V' \times V') \cap E$.

Two nodes v_i and v_j in a graph G are adjacent if an edge $e_{ij} = (v_i, v_j)$ exists in a graph. The neighboring information of a graph is represented by an adjacency matrix. The adjacency matrix $A \in \mathbb{R}^{|V| \times |V|}$ of a graph is defined as

$$A_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E, \\ 0 & \text{otherwise.} \end{cases} \tag{1.1}$$

The set of neighbourhood nodes with respect to a vertex v_i is denoted by $\mathcal{N}(v_i) = \{v_j \mid (v_i, v_j) \in E\}$. A degree or valency of node v_i in a graph G is the number of neighboring nodes, which is defined as

$$\deg(v_i) = |\mathcal{N}(v_i)|. \tag{1.2}$$

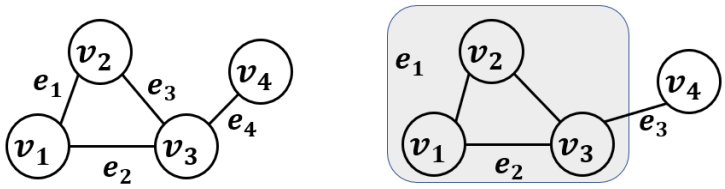


Figure 1.1: An undirected graph and an undirected hypergraph. **Left**: An undirected graph that has 4 vertices and 4 edges. **Right**: An undirected hypergraph that have 4 vertices and 3 edges. An edge e_1 is overlapped with half-toned. Edges e_1 and e_2 are duplication of vertices (v_1, v_3), but can have different edge labels.

A generalization of graphs is called a *hypergraph* $H = (X, E)$, where X is a set of vertices and E is hyper edges that have multiple edges rather than single pare of nodes (v_i, v_j). Examples of a graph and a hypergraph are shown in Fig. 1.1. The left graph in the figure shows a simple graph $G = (V, E)$ where a set of nodes and edges are given as $V = \{v_1, v_2, v_3, v_4\}$ and $E = \{e_1, e_2, e_3, e_4\}$, and each edge is a tuple: $e_1 = (v_1, v_2), e_2 = (v_1, v_3), e_3 = (v_2, v_3), e_4 = (v_3, v_4)$. The right graph in the figure shows a hypergraph $H = (X, E)$ defined as $X = \{v_1, v_2, v_3, v_4\}$, $E = \{e_1, e_2, e_3\}$, and each edge is a set: $e_1 = \{v_1, v_2, v_3\}, e_2 = \{v_1, v_3\}, e_3 = \{v_3, v_4\}$. Note that there are two more vertices in an edge e_1. This notation is convenient when considering graph generation by combining two subgraphs.

A graph can be decomposed into a junction tree composed of the set of subgraphs or cliques C. A junction tree $\mathcal{T}$ satisfies the following properties:

1. The union of all sets $C_1, \ldots, C_n$ equals to V; that is, $\bigcup_i C_i = V$.
2. For every edge $(u, v) \in E$, there exists $C_i \in \mathcal{V}$ such that $u \in C_i$ and $v \in C_i$.
3. If C_k is on a path from C_i to C_j in $\mathcal{T}$, $V_i \cap V_j \subseteq V_k$.

An example of junction tree is shown in Fig. 1.2. A graph G is decomposed into junction tree $\mathcal{T}$. Cliques in G include four cliques: $C_1 = \{v_1, v_2\}, C_2 = \{v_2, v_3, v_4\}$, $C_3 = \{v_3, v_4\}$, and $C_4 = \{v_4, v_5\}$. Intersection of vertices becomes edges of the original graph G in reconstruction from junction tree. These notation is helpful when computing marginalization in general graphs in Bayesian networks and generating graphs being retained with cycles.

1.3 Graph Kernels

The graph kernel is a kernel function that computes the similarity between graphs by the inner product of features computed from a graph. Many graph kernels

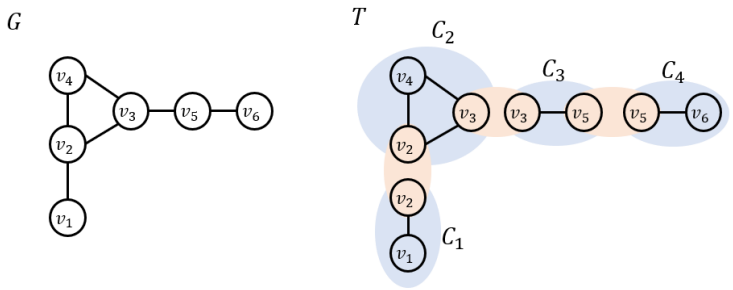


Figure 1.2: An undirected graph and its corresponding junction tree. **Left**: An undirected graph with a cycle that has 6 vertices and 6 edges. **Right**: A junction tree that has 4 cliques encircled with blue. Intersection in the junction tree is colored with orange. Graphs and trees can be converted interchangeably.

have been proposed to compute the similarity between graphs. A kernel function $k(x, x')$ is a measure of similarity between x and x' of features. Every kernel function must be symmetric about features, that is, $k(x, x') = k(x', x)$ and semi-definite [Vishwanathan et al., 2010]. The feature f computed from a graph is mainly composed of graph specific structures.

1.3.1 Vertex Histogram Kernel

One of the simplest kernels is the vertex histogram kernel that counts the occurrence of node labels in a graph G. Let d be the number of types of node labels, that is, $d = |\Sigma_v|$ and assume that $\Sigma_v = \{1, 2, \ldots, d\}$ without loss of generality. The histogram of node labels is $f_i = |\{v \in V : l_v = i\}|$. As a result the vertex histogram between G and G' is defined as

$$k_V(G, G') = \langle \mathbf{f}, \mathbf{f}' \rangle.$$

The complexity of the vertex histogram kernel is $O(|V|)$.

1.3.2 Weisfeiler–Lehman graph kernel

In order to take the subgraph structures into account, a number of efficient ways have been proposed. One of the powerful graph kernels is the Weisfeiler–Lehman subtree kernel [Shervashidze et al., 2011a], which is based on the Weisfeiler–Lehman test of isomorphism [Weisfeiler and Lehman, 1968]. Fig. 1.3 shows the procedure of computing Weisfeiler–Lehman algorithm at the second iteration with respect to v_0 and v'_0. Note that, for simplicity, the compression for relabeling is not shown in Fig 1.3. Each compressed label is determined after finishing iteration. The

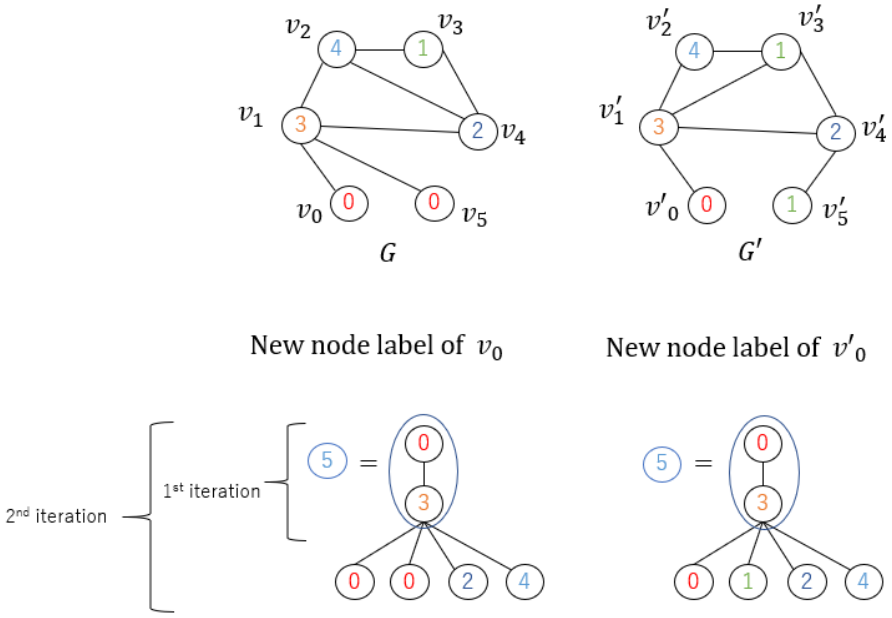


Figure 1.3: Weisfeiler–Lehman procedure. Given two graphs G and G', after second iteration, new labels of v_0 and v'_0 become different.

WL procedure is to aggregate node labels in the neighboring nodes and count its occurrence. For example, let us take the node label of v_0 and v'_0. In the first iteration, the neighboring node of v_0 is v_1, and they are aggregated as a new label $[0, 3]$. In order to compress the representation, the new node label is replaced as $5 := l([0, 3])$ in this case. The label of v'_0 is the same with that of v_0 at first iteration. After completing iteration over entire vertices, new labels in G and G' are shown in Fig 1.4.

A new label obtained by the WL procedure represents the sub-tree structure of a graph as shown in Fig 1.4. After iterating this process over and over, final substructures include duplication and are equal to original structure. At end of the iteration, feature vector representation is obtained as a series of counts of original node labels and counts of compressed node labels

$$k_{WL}(G, G') = \langle \phi(G), \phi(G') \rangle,$$

where ϕ is the vertex histogram of node labels after iteration.

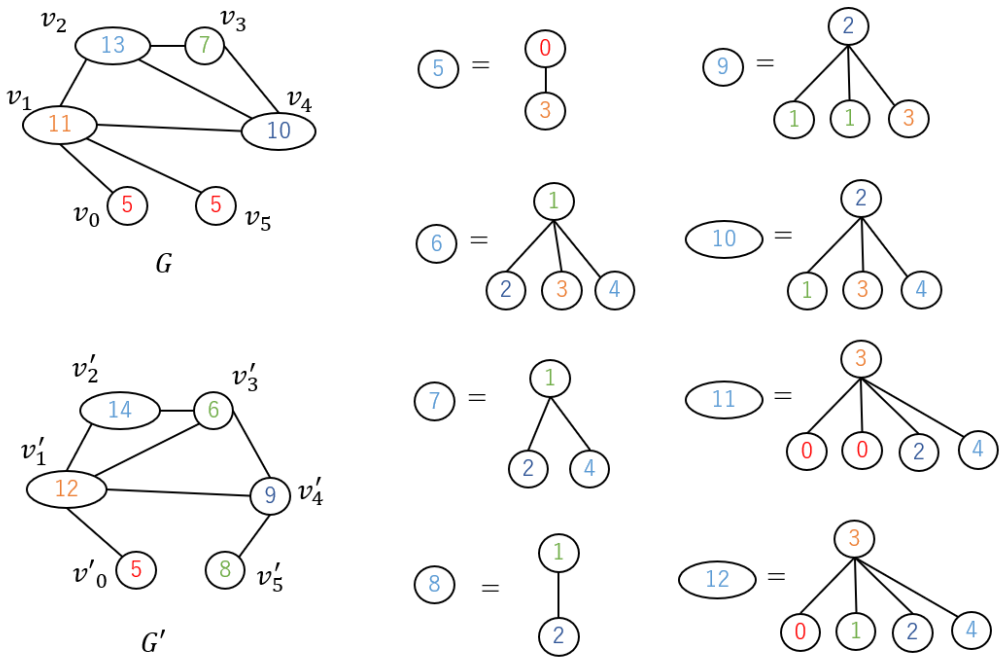


Figure 1.4: New node labels after 1st iteration in WL. These subgraphs are enumerated and counted in the end.

1.3.3 Extend connectivity Fingerprints

Extend connectivity fingerprints (ECFPs) are a novel class of topological fingerprints for molecular characterization [Rogers and Hahn, 2010a]. It is not a rigorous graph kernel, but it is highly related to the WL graph kernel procedure, so we state the algorithm in this section. In the section on Graph Neural Networks, we introduce the neural algorithm extended through graph neural networks.

The algorithm of ECFPs is basically the same process of aggregating neighboring node labels, such as atom labels. After obtaining a new node label, it is relabeled by a hash function that converts it to unique integer values to represent the subgraph structure. In ECFPs, received new labels are divided by the fixed length of the vector, the fingerprint S, and the remainder ($l_v \mod S$) is stored in the index of S as a bit that represents the existence of subgraph structures. The algorithm is shown in the left panel in Fig. 1.6. Counts of bits can be used instead of a bit as well. This algorithm can compute the fingerprint explicitly, but there is a problem of *bit collision* that an index of subgraph structures is duplicated. However, the subgraphs are different due to the modulo of fixed length.

1.4 Molecular Graphs and SMILES

Molecules can be defined as graphs. Notation of molecular graphs is mainly two ways. The first one is a string representation called Simplified molecular-input line-entry system, SMILES [Weininger, 1988]. Atoms are represented by the standard abbreviation of chemical elements such as carbon as C and nitrogen as N. Bonds are represented as the symbols, "=" is a double bound, "#" is a triple bond. Branching is defined in curly parenthesis "()' as in FC(F)F for fluoroform. Rings are represented as the number from start and end. For example, the compound benzene is written as c1ccccc1. If a chemical compound has multiple rings, such as naphthalene (that has two rings), the two numbers of closure 1 and 2 are needed to discern each ring, and it is written as c1c2ccccc2ccc1. SMILES can represent the same molecule in a different notation. In the case of ethanol, it may be written as CCO and OCC, but in canonical SMILES, ethanol is represented as CCO. Fig 1.5 shows the aspirin as the representation of a graph and SMILES. This SMILES is not canonical, but the starting point of the vertex begins in O (Oxygen). In canonical SMILES, aspirin is written as CC(=O)Oc1ccccc1C(=O)O. The one hot matrix of aspirin can be represented as illustrated in Fig 1.5, and this notation is helpful to generate SMILES by using a generative model.

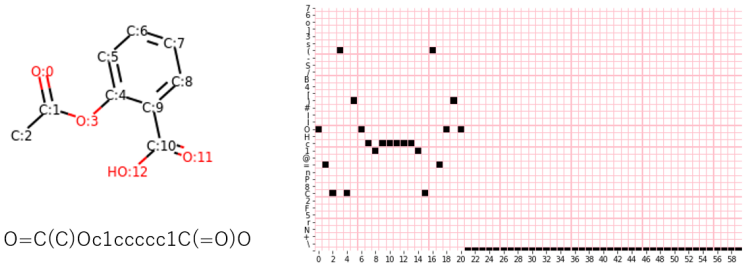


Figure 1.5: The notation of a graph and its SMILES of aspirin. **Left**: A molecular graph and its SMILES. **Right**: The lookup table for SMILES. The vertical axis shows the characters in SMILES, and the horizontal axis shows the position of SMILES.

1.5 Graph Neural Networks

The first concept of graph neural networks (GNNs) are originally introduced by [Gori et al., 2005, Scarselli et al., 2009] extending recursive neural networks into a graphs to expressing topological information of graphs. The recursive definition of node and edge features is adopted to update them through parametric functions. The first GNNs have some drawbacks about training properly that the function must be a "contraction map" and many iterations are need to reach a stable state [Zhang et al., 2018]. In order to overcome this problem, an improvement of GNNs is gated graph sequence neural networks [Li et al., 2015].

Message passing neural networks are unified forms of most of proposed GNN variants [Gilmer et al., 2017]. The message passing scheme is basically equivalent to the WL scheme, that is, it aggregates node features of neighboring nodes. After aggregating node features, these features are updated as a new message $m_v^{(t)}$ for a node v through the message function M_t with edge features e_{vw} at the tth WL iteration round. Hidden features h_v are updated through the vertex update function U_t. These procedure is defined as

$$m_v^{(t+1)} = \sum_{w \in \mathcal{N}(v)} M_t(h_v^{(t)}, h_w^{(t)}, e_{vw}), \tag{1.3}$$

$$h_v^{(t+1)} = U_t(h_v^{(t)}, m_v^{(t+1)}), \tag{1.4}$$

where, in the sum, $\mathcal{N}(v)$ denotes the neighbor of v in a graph G. The readout phase to extract graph features at the final round of message passing T is defined as follows:

$$\hat{y} = R(\{h_v^{(T)} | v \in G\}), \tag{1.5}$$

where R is some readout function that can be learnable differentiable function. Most proposed GNNs can be represented as the message passing scheme. For example,

Algorithm 1 Circular fingerprints

Input: molecule, radius R, fingerprint length S
Initialize: fingerprint vector $\mathbf{f} \leftarrow \mathbf{0}_S$
for each atom a in molecule
  $\mathbf{r}_a \leftarrow g(a)$ ▷ lookup atom features
for $L = 1$ to R ▷ for each layer
  for each atom a in molecule
    $\mathbf{r}_1 \dots \mathbf{r}_N = \text{neighbors}(a)$
    $\mathbf{v} \leftarrow [\mathbf{r}_a, \mathbf{r}_1, \dots, \mathbf{r}_N]$ ▷ concatenate
    $\mathbf{r}_a \leftarrow \text{hash}(\mathbf{v})$ ▷ hash function
    $i \leftarrow \text{mod}(r_a, S)$ ▷ convert to index
    $\mathbf{f}_i \leftarrow 1$ ▷ Write 1 at index
Return: binary vector $\mathbf{f}$

Algorithm 2 Neural graph fingerprints

Input: molecule, radius R, hidden weights $H_1^1 \dots H_R^5$, output weights $W_1 \dots W_R$
Initialize: fingerprint vector $\mathbf{f} \leftarrow \mathbf{0}_S$
for each atom a in molecule
  $\mathbf{r}_a \leftarrow g(a)$ ▷ lookup atom features
for $L = 1$ to R ▷ for each layer
  for each atom a in molecule
    $\mathbf{r}_1 \dots \mathbf{r}_N = \text{neighbors}(a)$
    $\mathbf{v} \leftarrow \mathbf{r}_a + \sum_{i=1}^{N} \mathbf{r}_i$ ▷ sum
    $\mathbf{r}_a \leftarrow \sigma(\mathbf{v} H_L^N)$ ▷ smooth function
    $\mathbf{i} \leftarrow \text{softmax}(\mathbf{r}_a W_L)$ ▷ sparsify
    $\mathbf{f} \leftarrow \mathbf{f} + \mathbf{i}$ ▷ add to fingerprint
Return: real-valued vector $\mathbf{f}$

Figure 1.6: Circular fingerprints and Neural graph fingerprints (source: [Rogers and Hahn, 2010a]). **Left**: The algorithm of Circular Fingerprints (ECFPs). The main difference of WL is after mod operation. **Right**: Neural graph fingerprints use a nonlinear function to embed subgraphs into latent spaces.

Neural Graph Fingerprints [Duvenaud et al., 2015] are related to the procedure of the circular fingerprints and WL. The difference is shown in 1.6. Main difference between circular fingerprints and neural graph fingerprints is to embed aggregated node labels into vectors instead of hashing. After that, the features are updated through the sigmoid function and allocated the index of vectors. Neural graph fingerprints can be defined as the message passing framework in the following.

$$M(h_v, h_w, e_{wv}) = \text{CONCAT}(h_w, e_{vw}), \tag{1.6}$$

$$U_t(h_v^{(t)}, m_v^{(t+1)}) = \sigma(H_t^{\deg(v)} m_v^{(t+1)}), \tag{1.7}$$

$$R = f\left(\sum_{v,t} \text{softmax}(W_t h_t^{(t)})\right), \tag{1.8}$$

where σ is the sigmoid function, H_t is a learned matrix for each timestep t, and R has a skip connections to all previous hidden states $h_v^{(t)}$.

Recent graph neural networks are unified with the graph networks [Battaglia et al., 2018]. These GN frameworks are based on building blocks of functions: an "update" function ϕ and three "aggregation" functions, ρ. Let the graph features be $\mathbf{u}$. Three update functions are defined as follows:

$$\mathbf{e}'_k = \phi^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}), \tag{1.9}$$

$$\mathbf{v}'_k = \phi^v(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u}), \tag{1.10}$$

$$\mathbf{u}' = \phi^u(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u}). \tag{1.11}$$

Aggregation functions are defined as

$$\bar{\mathbf{e}}'_i = \rho^{e\to v}(E'_i), \tag{1.12}$$
$$\bar{\mathbf{e}}' = \rho^{e\to u}(E'), \tag{1.13}$$
$$\bar{\mathbf{v}}' = \rho^{v\to u}(V'), \tag{1.14}$$

where

$$E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{r_k=i,k=1:|E|}, \tag{1.15}$$
$$V' = \{\mathbf{v}'_i\}_{i=1:|V|}, \tag{1.16}$$
$$E' = \cup_i E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{k=1:|E|}. \tag{1.17}$$

r_k is the index of the receiver node, and s_k is the index of the sender node. The ϕ^e is mapped across all edges, ϕ^v is mapped across all nodes, and ϕ^u is mapped across the entire graph features to compute update the features obtained from aggregation functions, respectively. The ρ function aggregates each element and reduces it to a single element as a message. The ρ function must be invariant to permutations; examples of aggregation functions are summation, mean, maximum etc.

1.6 Reinforcement Learning

Machine Learning has three topics: Supervised learning, Unsupervised learning, and Reinforcement Learning. In reinforcement learning, the core idea is to make an agent learning the optimal behavior so as to maximize a numerical reward through interacting with the environment as shown in Fig. 1.7. The agent interacts an environment at $t = 0, 1, \ldots$. At each time step t, the agent receives a state s_t and an immediate reward r_t. Based on some policy computed from the current state, the agent takes the action a_t. As a result, the environment returns the new state s_{t+1} and the numerical reward r_{t+1}. Through this agent-environment interaction, the agent trains the optimal action so as to maximize the cumulative rewards.

The agent's policy is denoted as $\pi(a_t|s_t)$, which is the probability of taking an action a_t given the state s_t. The agents tries to compute actions so as to maximize the expected reward defined as

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1},$$

where γ ($0 \le \gamma \le 1$) is a discount rate to control how the future reward is weighted. If γ is equal to zero, agents take an action based on the immediate reward, not

Figure 1.7: The diagram of environment interaction in reinforcement learning.

considering the future rewards. As γ approaches 1, the agents take future rewards into account.

Reinforcement learning has been applied to various domains such as game, robotics, simulation, and chemo informatics. As for molecular graph generation, model-free and model-based algorithms can be considered because if molecular graphs are treated as discrete structures instead of graph embedding, model-based algorithms such as MCTS can be applied to search directly graph structure so as to satisfy with target properties. In contrast, if graphs are embedded and generated through graph neural networks and generative models, then model free algorithms such as Deep Q Network, actor critic and Proximal Policy Optimization can be applied to search the latent space to generate graphs. In such cases, other optimization techniques such as Bayesian optimization and meta-heuristics can be applied to search the latent space as well.

1.6.1 Policy Gradient

Policy gradient methods are computed by differentiating the objective

$$L^{\mathrm{PG}}(\theta) = \hat{\mathbb{E}}_t[\log \pi_\theta(a_t|s_t)\hat{A}_t],$$

where π_θ is a stochastic policy function and $\hat{A}_t$ is an advantage estimator at time step t. The expectation $\hat{\mathbb{E}}_t$ indicates the empirical average over a finite batch of samples. A generalized advantage estimator is computed from exponentially-weighted discounted cumulative average over a reward sequence [Schulman et al., 2015b]. This algorithm is used for continuous control.

1.6.2 Proximal Policy Optimization

Proximal policy optimization (PPO) is based on trust region policy optimization (TRPO) methods [Schulman et al., 2015a] to prevent the high variance while

learning in policy gradient [Schulman et al., 2017]. The key idea of PPO is using the clipped surrogate objective as the following:

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right]$$
$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)},$$

where ϵ is a clip parameter. The first term inside the min is conservative policy iteration [Kakade and Langford, 2002], and the parameter θ of policy is updated through mini-batch stochastic gradient ascent. The second term modifies the surrogate objective by clipping the probability ratio, which removes the incentive for moving r_t outside of the interval $[1-\epsilon, 1+\epsilon]$ [Schulman et al., 2017]. The PPO also exploits the value function for estimating the cumulative rewards, and the entropy term for exploration. In the end, the objective considering the value function and exploration is defined as follow:

$$L_t^{\text{CLIP+VF+S}}(\theta) = \hat{\mathbb{E}}_t \left[L_t^{\text{CLIP}}(\theta) - c_1 L_t^{\text{VF}}(\theta) + c_2 S[\pi_\theta](s_t) \right]$$

where, c_1, c_2 are coefficients, and S denotes the entropy bonus. L_t^{VF} is squared loss of $V_\theta(s_t)$ and the target value function.

1.7 Generative Models for Molecular Graph Generation

In this section, we state the fundamental molecular graph generation model using Variation Autoencoder and Reinforcement Learning. Later in Chapter 2, we propose a new method to generate novel molecular graphs using sub-graph mining methods, Monte Carlo Tree Search (MCTS), and reinforcement Learning.

Chemical VAE Chemical VAE is proposed to generate SMILES using a variational autoencoder(VAE) [Gómez-Bombarelli et al., 2018]. SMILES representation is string forms of a molecule. In stead of generating molecules directly, SMILES can be generated by Chemical VAE. In order to generate SMILES with desired properties, neural networks are trained jointly with multilayer perceptrons to predict a desired property values through the continuous representation of molecules computed from an encoder. After training the generative model, the latent vectors of molecules with a target property are searched by Bayesian optimization. The optimized latent vectors are decoded to generate characters of SMILES, and they are reconstructed into SMILES. However, reconstructing SMILES from characters is normally highly challenging because the representation pattern of SMILES is difficult to learn the generalization without considering SMILES grammar rules.

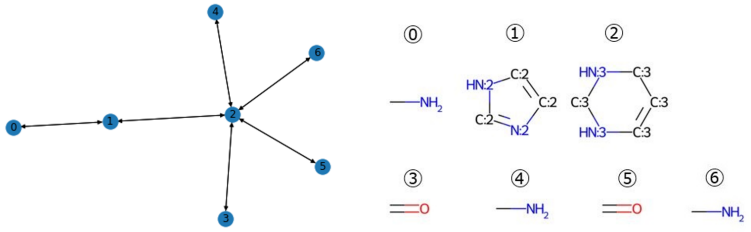


Figure 1.8: A junction tree generated by VAE. Left figure shows the generated junction tree. Right figure shows that the number of each node corresponds to substructures.

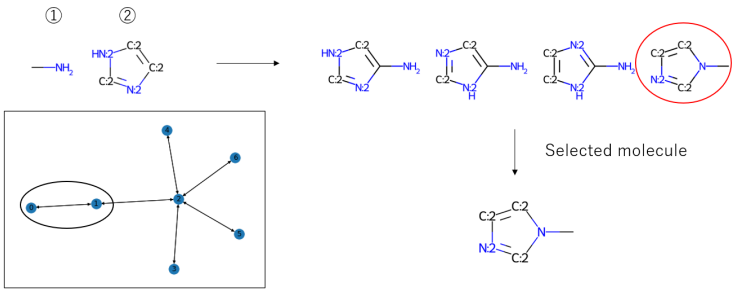


Figure 1.9: An example of assembling process. The label encoder outputs the log-likelihood of reassembled molecules.

In order to address this problem, the VAE model that generates a grammar rule instead of characters is proposed [Kusner et al., 2017].

Junction Tree VAE Generating SMILES has another problem when generating rings. Valid SMILES must be grammar rules. If the character of end points of rings shifts even by one, the entire SMILES would be invalid. In Fig. 1.5, for example, in order to reconstruct SMILES from a one hot table, the position of "1" must be exacted. Junction tree VAE [Jin et al., 2018] is proposed to generate junction trees and reconstruct molecules by using VAE. The model is composed of two models: the tree encoder model and the label encoder model in order to overcome the problem of invalid generating SMILES. Fig. 1.8 shows the generating process of JT-VAE. The tree autoencoder generates the structure of a junction tree and its node labels as substructures. The label encoder, which is message passing networks), trains how subgraph structures (cliques obtained from a junction tree) are combined, and selects which subgraphs are assembled according to the likelihood shown in Fig 1.9. In order to generate molecules with desired properties, two latent vectors in tree encoders and the label encoder need to be searched.

GCPN Graph Convolutional Policy Network for Goal-Directed Molecular

Graph Generation (GCPN)[You et al., 2018] is the method to generate graphs in a node-by-node manner with reinforcement learning. The state of a graph is computed from graph convolutional networks. Action space is defined as a tuple of (start node id, end node id, edge type and stop signal). These actions are computed from MLPs and the policy is optimized by PPO.

1.8 Our Contributions

Generating molecular graphs with desired properties have mainly several challenging tasks to tackle, as follows:

1. How should molecular graphs be represented and reconstructed from embedding features into the original?
2. How should graph properties, such as whether molecules are toxic or not, be predicted?
3. How should the molecules with desired properties searched in huge chemical space?

In Chapter 2, in order to solve the problems of 1 and 3, we propose a new graph generation algorithm called MOLDR (Molecular graph Decomposition and Reassembling). This algorithm has two steps and can generate molecules directly instead of generative models. In the first decomposition step, subgraph structures are mined by the subgraphs mining algorithm. In the second step, desirable graphs are searched via Monte Carlo Tree Search and reinforcement learning. This procedure of decomposition and reassembling enables interpretable graph generation. The advantage of our method MOLDR is summarized as follows:

- Our method MOLDR explicitly constructs new molecules by combining substructures of molecules instead of using generative models; hence, its generation process is interpretable.
- MOLDR can easily generate larger molecules out of distribution in a dataset by combining subgraph structures.
- Molecules generated by MOLDR are superior to those by the current state-of-the-art methods in terms of $\log P$ and QED (drug-likeness).

Next, in Chapter 3, in order to guide the target graph with desired properties, an appropriate prediction function for the target is needed, and graph neural networks are often used for this purpose. However, GNNs using a message passing scheme

have a problem with over-smoothing. When over-smoothing happens, the similarity between node features is close to each other, and the effect of a large amount of subgraph structures cannot be taken into account.

Considering the larger graphs in the process of generating graphs, in order to evaluate the problem of 2, we investigate the prediction performance of graph machine learning and the graph features as the number of message passing iterations increases. As a result, we gain insights into the prediction performance of graph neural networks as below:

- We show that the prediction performance of the WL kernel outperforms the most fundamental GNNs, GCNs, and GINs if the number of message passing iterations increases.
- The WL kernel does not significantly deteriorate for many message passing iterations in most datasets. At the same time, GCNs and GINs do due to their large number of parameters and ill-trained previous node features.
- We also show that the transition of node features tends to be small, which leads to the difficulty of determining which feature is informative for prediction. The features become certain values when the training fails.

In later chapter, we will state our methods and the results in detail.

Chapter 2

Molecular Graph Generation

2.1 Introduction

Designing new molecules for drug and material with desired properties is a challenging task due to the massive number of potential drug-like molecules, which is estimated to be between 10^{23} to 10^{60} [Polishchuk et al., 2013]. Molecules are essentially represented as *graphs* with node and edge attributes, while such graph structure of chemical compounds makes it difficult to generate valid molecules with desired activity or property even if you can build a Quantitative Structure-Activity Relationship (QSAR) model, which is a computational modeling method for revealing relationships between structural properties of chemical compounds and biological activities [Kwon et al., 2019], by designing descriptors of chemical features specifically for virtual screening. The straightforward way of generating molecules is to solve the inverse QSAR problem through the objective function estimated from the molecular structures [Miyao et al., 2016, Wong and Burkowski, 2009, Churchwell et al., 2004]. However, feature vectors extracted from molecular graphs are often highly correlated between its features, which makes it challenging to reconstruct a new molecular graph from the optimized descriptors as it requires preserving such correlation information.

A number of methods have been proposed to tackle this problem of molecular generation [Gugisch et al., 2014, Takeda et al., 2020, Olivecrona et al., 2017]. Recent advanced approaches to finding of drug-candidate molecules have employed deep generative models [Gómez-Bombarelli et al., 2018, Kusner et al., 2017, Guimaraes et al., 2017, Jin et al., 2018]. The basic idea of using generative models is to learn the latent representation of molecules, which enable us to reconstruct and explore molecules that satisfy target properties in the learned latent chemical space. Exploration methods such as Bayesian optimization is used to search the latent chemical space [Gómez-Bombarelli et al., 2018]. However, it is

fundamentally difficult to reconstruct molecular graphs from the latent space and to search molecules with the desired property by extrapolation from a training dataset as a large part of the latent space represents invalid molecules.

Another strategy to search desired molecules is based on a reinforcement learning. In the setting of reinforcement learning, an agent learns the optimal policy to maximize the cumulative reward, and the trained agent can take an action to generate the optimal molecules. The recurrent neural network (RNN) model can be used to generate a string notation of a molecule, which is a simplified molecular-input line-entry system (SMILES) [Weininger, 1988]. The agent takes an action of the next character of SMILES based on the optimized policy [5]. In the case of molecular graph generation using reinforcement learning, the agent takes an action of choosing the atom type and bond type between nodes to expand each molecule [You et al., 2018]. The state is represented as the latent feature vectors by using RNNs or graph neural networks. However, both approaches of SMILES generation and node-wise molecular graph generation share the problem that the intermediate steps do not represent valid molecules, which significantly deteriorates the interpretability of resulting generated molecules. Moreover, the property and the state radically change if the ring structure appears, and it is fundamentally difficult to treat such binary response in optimization on a continuous latent space.

In this chapter, we propose a novel molecular generation approach, called MOLDR (MOLecular graph Decomposition and Reassembling), which generates optimized new molecules by decomposing molecular graphs into subgraphs in a training dataset and reassembling such obtained subgraphs again in a different way. Our key insight is that chemical properties depend on the combination of subgraphs, which correspond to the functional group or the motif of molecules in the context of chemoinformatics, and that it can be optimized when appropriate substructures are included in molecules. MOLDR is composed of a *decomposition* step and a *reassembling* step. In the decomposition step, we first convert each molecular graph into a tree structure to efficiently obtain subgraphs; that is, functional groups, followed by extracting frequent subgraph structures by applying a graph mining method. In the reassembling step, we treat the extracted subgraphs as building blocks of molecular graphs and reassemble them by searching desired blocks according to the target property using Monte Carlo tree search (MCTS) [Coulom, 2006, Kocsis and Szepesvári, 2006]. We empirically evaluate molecular graphs generated by our method with respect to two well-established property scores, the penalized log P and Quantitative Estimation of Drug-likeness (QED) [Bickerton et al., 2012], and show that our method outperforms the state-of-the-art molecular generation methods.

Our contributions are summarized as follows:

- Our method MOLDR explicitly constructs new molecules by combining substructures of molecules, hence its generation process is interpretable.
- MOLDR can easily generate larger size of molecules out of distribution in a dataset by combining subgraph structures.
- Molecules generated by MOLDR are superior to those by the current state-of-the-art methods in terms of $\log P$ and QED (drug-likeness).

2.2 Related works

[Yang et al., 2017, 2021] and [Olivecrona et al., 2017] proposed a SMILES generation approaches by RNNs and search the SMILES with desired properties using MCTS and policy gradient respectively. Instead of generating SMILES, [You et al., 2018] proposed node-wise graph generation and property optimization using reinforcement learning. The state of a molecule is represented through graph convolutional networks, and the agent selects nodes, edges types, or terminal to expand molecules. The policy is optimized through the Proximal Policy Optimization. These methods can generate valid molecules with desired properties at the final step. However, the generation process is a black-box and it is difficult to explain why and how such molecules are obtained.

[Jin et al., 2018] proposed a VAE model that generates junction trees over molecules. Nodes in a junction tree represent subgraphs extracted from a molecular dataset, and a graph neural network determines which nodes or edges are combined with each other in the junction tree. To search molecules that optimize the desirable properties, it is necessary to search two vectors, what a tree structured scaffold is and how a molecule is reconstructed within latent embedding space. In contrast, in our method, junction trees themselves are used to efficiently extract frequent substructures from a molecular dataset, and we expand and search substructures directly to achieve target scores instead of generating junction trees from VAE.

[Takeda et al., 2020] proposed to generate molecules by combining substructures which contribute to the target properties, where candidate molecules are searched by McKay's Canonical Construction Path (MC-MCCP) algorithm [McKay, 1998, Stephen and Andrew, 2009]. [Jin et al., 2020] proposed the multi objective molecule generation using interpretable substructures as rational for extracting substructures by MCTS to generate molecules by merging common substructures and graph completion. Although their approaches and our approach share the general strategy of constructing new molecules from its substructures, our method can cover a wider variety of substructures in molecular generation as we directly apply frequent

subgraph mining to the entire molecular dataset, which will lead to better new molecules.

Our approach of combining decomposition of molecules into subgraphs by graph mining and reassembling of subgraphs to generate new molecular graph has not been studied at sufficient depth. There is a related approach in the task of planning of chemical synthesis, which also combines subgraphs and MCTS [Segler et al., 2018b], while it is not applied to the property optimization.

2.3 The Proposed Algorithm: MOLDR

We introduce our molecular generation algorithm MOLDR. We provide the problem setting, tree decomposition preprocessing, graph mining, and the strategy to build up molecules via Monte Carlo Tree Search and Proximal Policy Optimization.

2.3.1 Problem Setting

A *graph* is a tuple $G = (V, E)$, where V and E denote the set of nodes and edges, respectively. Nodes and edges can have labels (attributes) via label functions $l_V : V \to \Sigma_V$ for nodes and $l_E : E \to \Sigma_E$ for edges with some label domain Σ_V, Σ_E, which can be any set such as $\mathbb{Z}$ and $\mathbb{R}^d$. We assume that each molecule is represented as a graph. If we see a graph as a molecule, V is the set of atom types, and E is the set of bond types. For two graphs $G = (V, E)$ and $G' = (V', E')$, we say that G' is a *subgraph* of G, denoted by $G' \sqsubseteq G$, if $V' \subseteq V$ and $E' \subseteq (V' \times V') \cap E$.

Let us assume that $f(G)$ is some chemical property of a graph G, which is usually a real-valued function, where we assume that f is known beforehand and we can compute $f(G)$ for any graph G. For example, f can be the log P of a molecular G. Given a molecular dataset, which is a collection of graphs, the problem of molecular generation is to explore a new graph G_{new} that has high $f(G_{\text{new}})$ value as long as possible.

2.3.2 Graph Decomposition via Frequent Subgraph Mining

Given a collection of graphs as an input molecular dataset, our idea is to apply frequent subgraph mining [Zaki and Meira, 2014] to the dataset, which finds subgraphs that frequently appear in in the graph dataset. Formally, given a graph dataset $\mathbf{D} = \{G_1, G_2, ..., G_n\}$, the objective of frequent subgraph mining is to find all subgraphs G satisfying the condition $\text{support}(G) \geq \text{minsup}$, where $\text{support}(G)$ is defined as

$$\text{support}(G) = \Big| \{G_i \in \mathbf{D} \mid G \sqsubseteq G_i\} \Big|,$$

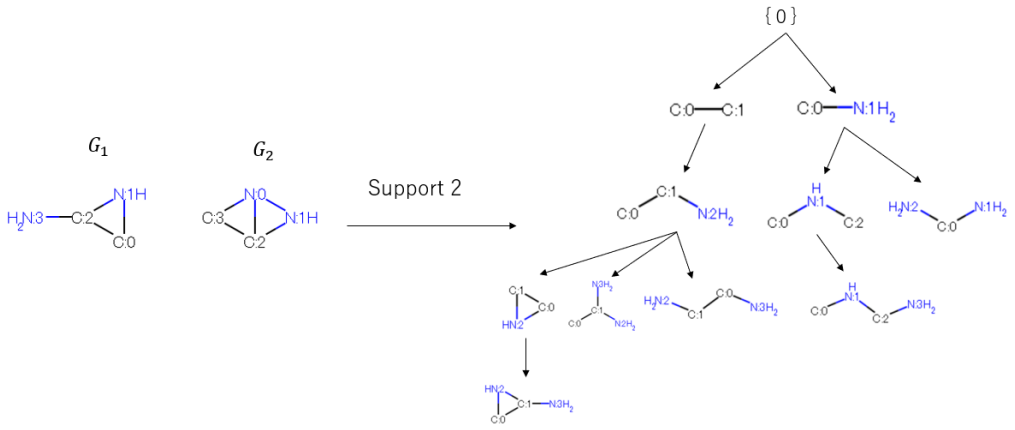


Figure 2.1: An example of gSpan applied to two molecular graphs under support 2. In support 2, all subgraphs are enumerated in a lexicographical order.

that is, the number of graphs in **D** that contains G as a subgraph, and minsup $\in \mathbb{Z}$ is a frequency threshold.

We use the gSpan [Xifeng Yan, 2002] algorithm, which is commonly used for the task of frequent subgraph mining. It enumerates subgraphs in a depth first manner. Each graph is represented as the so-called DFS code, which is constructed from a search tree based on a lexicographic order and enables us to efficiently check duplication of enumerated graphs as shown in Fig 2.1. More precisely, for each explored graph during the enumeration, it checks whether or not its DFS code is canonical. After completion of gSpan, we check every enumerated subgraph and keep only subgraphs whose target property score is already higher than some threshold, which is determined beforehand, to efficiently reassemble them to construct new graphs in the next reassembling step.

Molecules are firstly converted into molecular graphs, where each node represents an atom type and each edge represents a bond type. However, if we directly apply gSpan to such molecular graphs, it gives a lot of invalid subgraphs in terms of molecules as building blocks for molecular generation as shown in Fig 2.2. This is because gSpan does not know the chemical context and simply enumerates frequent subgraphs, hence, for example, the ring structure will be truncated by gSpan, while such truncated subgraphs are invalid and unnecessary for the reassembling step.

To circumvent this problem, we apply *tree decomposition* to molecular graphs as preprocessing before applying gSpan, and convert them into molecular junction trees. A tree decomposition maps a graph $G = (V, E)$ into a *junction tree* $\mathcal{T} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{C_1, \ldots, C_n\}$ is a collection of subsets of V; that is, each $C_i \subseteq V$, and $\mathcal{E}$ is a set of edges between elements of $\mathcal{V}$. A junction tree satisfies

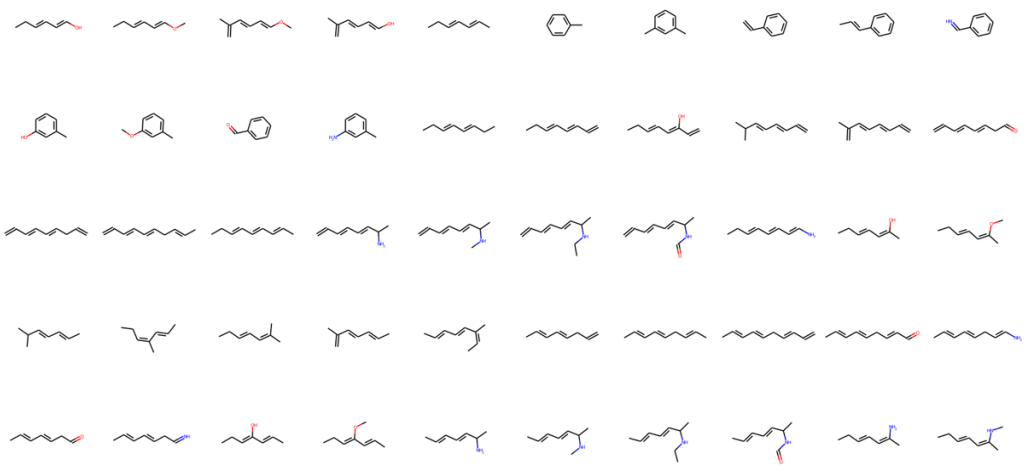


Figure 2.2: An example of directly applying gSpan to molecular graphs. The enumerated subgraphs include truncated structures of molecules. Chemical properties such as ring of benzene are completely ignored and more graphs are mined through gSpan.

the following properties:

1. The union of all sets $C_1, \ldots, C_n$ equals to V; that is, $\bigcup_i C_i = V$.
2. For every edge $(u, v) \in E$, there exists $C_i \in \mathcal{V}$ such that $u \in C_i$ and $v \in C_i$.
3. If C_k is on a path from C_i to C_j in $\mathcal{T}$, $V_i \cap V_j \subseteq V_k$.

By converting a graph into its corresponding junction tree, by definition, each cycle will be gathered as a single node and all cycles will be eliminated. Therefore, if we apply gSpan to not the original graphs but the converted junction trees, we can avoid enumerating invalid subgraphs in which the ring structure of a molecule, represented as a cycle on a graph, is truncated. In addition, gSpan on junction trees can dramatically reduce the number of frequent subgraphs. This is also an advantage of using junction trees in the decomposition step for molecular generation.

The edge label information and the node label information in each clique are lost in a junction tree, hence we need to restore them after frequent subgraph mining. To achieve this task, we use a subgraph matching algorithm that matches between original graphs and obtained trees. We use the indexed based subgraph matching algorithm with general symmetries (ISMAGS) [Houbraken et al., 2014]. Since the size of each molecule is usually not so large and the number of nodes is mostly around 20~30 in the task of molecular generation, this restoring process is not computationally expensive.

2.3.3 Graph Reassembling from Frequent Subgraphs

Now we generate new molecules by reassembling frequent subgraphs obtained by the previous graph decomposition step. In contrast to our approach using subgraphs as building blocks, existing approaches are based in either text generation or node-wise graph generation. In the text generation approach [Segler et al., 2018a] based on SMILES, an algorithm picks up a particular character which denotes chemical state, such as the atom (C, N, O, F, ...), the bond type (=, ≡), or the branched symbols, from the set of character types occurred in a training dataset to generate and expand molecules. In the node-wise graph generation [You et al., 2018, Li et al., 2018], an algorithm selects a node (atom symbol) and the edge type between source and target atoms from the candidate set of atom and edge types. Our method can be more powerful as we directly combine subgraphs that already have desirable properties as building blocks in molecular generation.

To assemble molecular subgraphs, we pick up two graphs G_t and G'_t from building blocks and combine them to generate a new graph G_{t+1}, where t is the number of building up steps of molecules. Let us assume that $G_t = (V(G_t), E(G_t))$ with $V(G_t) = \{v_1, \ldots, v_n\}$ and $G'_t = (V(G'_t), E(G'_t))$ with $V(G'_t) = \{u_1, \ldots, u_{n'}\}$. In the reassembling with nodes, we select single nodes $v_i \in V(G_t)$ and $u_j \in V(G'_t)$ such that they have the same node labels: $l_v(v_i) = l_v(u_j)$. We overlay these two nodes as v_{t+1}; that is, $V(G_{t+1}) = V(G_t) \cup V(G'_t) \setminus \{v_i, u_j\} \cup \{v_{t+1}\}$ for a newly constructed graph G_{t+1}. All edges in G_t and G'_t are preserved in G_{t+1}, where if there is an edge (v_i, v_k) or (u_j, u_l), it is replaced with (v_{t+1}, v_k) or (v_{t+1}, u_l). In the reassembling with edges, we select edges from rings, and overlay them in the same manner as the assembling with edges. This assembling is similar to reconstructing a graph from a junction tree; that is, nodes of a clique in a junction tree have intersected nodes that are connected with each other between subgraphs. Assembling two graphs is equivalent to choose the intersection of nodes or edges. Figure 2.3 shows the process of reassembling the two molecular graphs with nodes or edges, respectively. The candidate set of node label C for merging is $\{v_0 : \{u_6, u_7, u_9, u_{10}, u_{15}\}, v_4 : \{u_6, u_7, u_9, u_{10}, u_{15}\}, v_5 : \{u_6, u_7, u_9, u_{10}, u_{15}\}\}$, where indices of nodes correspond to numbers in the illustration in Figure 2.3. We do not include internal nodes such as u_{12} as the resulting graph will be an invalid molecule. In the reassembling with edges, the candidate set in the same node label (C, C) is $\{\{(v_0, v_1), (u_6, u_7)\}, \{(v_1, v_2), (u_6, u_7)\}\}$, resulting in two new graphs.

The computational cost of combining two graphs depends on the number of nodes and the number of edges in rings. In the worst case, where we need to consider all combinations of nodes and edge of two graphs $G = (V, E)$ and $G' = (V', E')$, the complexity becomes $O(|V||V'| + |E||E'|)$. However, this can be usually reduced in practice by considering the symmetrical structure of a graph and some type of restrictions of chemical valency of an element in the case of molecules. We always

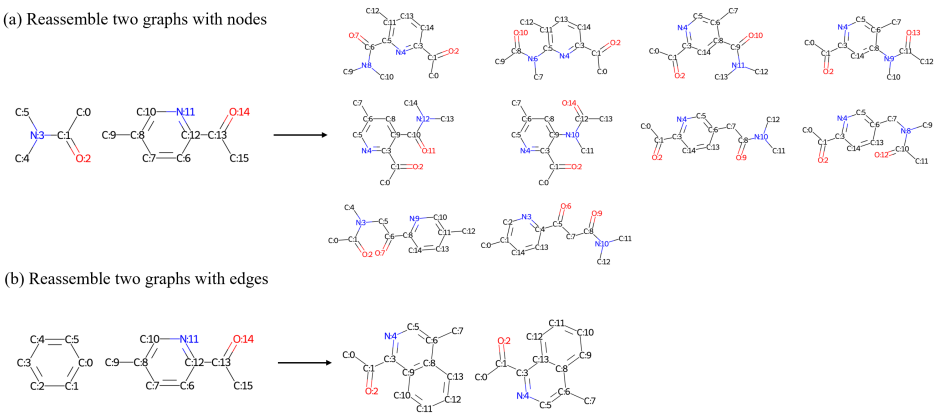


Figure 2.3: Reassembling two molecules with nodes (a) or edges (b). (a) shows merging with nodes labeled as C. (b) shows merging with edges in rings. In this examples, reassembled molecules are sanitized to be valid molecules.

check such conditions whenever a new molecule is generated and remove it if does not satisfy such conditions. Therefore molecules generated by our method is always valid.

2.3.4 Finding Candidate Subgraphs by Monte Carlo tree search

To efficiently find which graphs should be assembled in graph generation, we use Monte Carlo tree search (MCTS), which is a search method that combines tree search with random sampling [Browne et al., 2012]. MCTS has been applied to a number of tasks when the search space is massive and achieved huge success in various fields such as the game of Go [Silver et al., 2016] and the planning of chemical syntheses [Segler et al., 2018b].

Forward pass. The first node can be any subgraph for molecular graph generation. The search stops when the number of steps reaches at the maximum number of steps L or a the weight of a generated molecule exceeds the predetermined threshold. At each time step $t < L$, an action for expanding a search tree from a current molecule G_t is selected according to the following functions:

$$A_t = \underset{A}{\operatorname{argmax}}\big(Q(G_t, A) + U(G_t, A) \big), \tag{2.1}$$

$$Q(G_t, A) = \frac{W(G_t, A)}{N(G_t, A)}, \quad U(G_t, A) = c_{\text{pcut}} R(G_t, A) \frac{\sqrt{\sum_B N(G_t, B)}}{1 + N(G_t, A)} \tag{2.2}$$

where A is a candidate subgraph from building blocks, $N(G_t, A)$ is a visit count of the pass, $W(G_t, A)$ is the score of the chemical property, $R(G_t, A)$ is the predicted

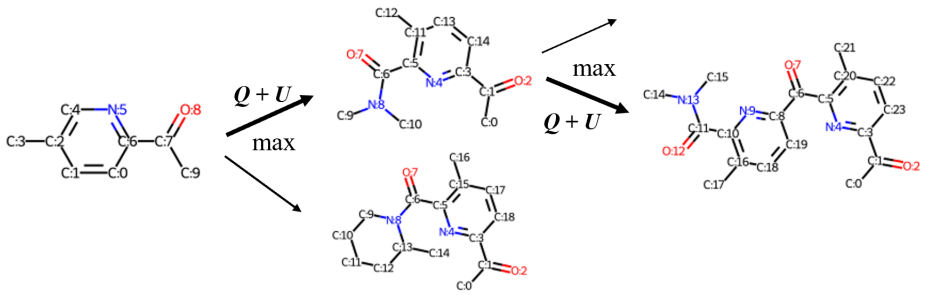


Figure 2.4: The molecular generation via MCTS. Subgraph structures are selected based on the PUCT scores in Equation (2.1). After reassembling molecules, a new graph is selected based on the highest score of a target property.

target property score of a new graph G_{t+1} assembled from previous node and a selected building block, and c_{pcut} is a constant value that determines the level of exploration. The function $Q(G, a)$ represents the mean action value. This algorithm is a variant to predictive upper confidence tree (PUCT) [Rosin, 2011].

Backward pass. The rollout statistics are updated for each step as follows:

$$N(G_t, A_t) := N(G_t, A_t) + 1, \tag{2.3}$$

$$W(G_t, A_t) := W(G_t, A_t) + R(G_{\text{leaf}}), \tag{2.4}$$

where $R(G_{\text{leaf}})$ is the score of the terminal molecule G_{leaf}. We collect all of the reassembled molecules through these steps. Figure 2.4 shows the procedure of MCTS. The building blocks; that is, subgraphs extracted by frequent graph mining, are selected based on $\max(Q(G_t, A) + U(G_t, A))$. When graphs are reassembled, several candidates of new graphs are generated. We prune such new graphs by considering molecules with the higher score according to the target property. This procedure proceeds until the terminal condition is satisfied, such as the maximum number of nodes or the molecular weight.

2.4 Experiments

We empirically examine the effectiveness of our proposed method MOLDR compared to the state-of-the-art molecular generation methods. In particular, we have examined the two standard criteria, the pernalized $\log P$ and the drug-likeness score QED, of generated molecules. In addition, we also examine the multi-objective score of QED and SA. Furthermore, we benchmark the rediscovery molecules using GuacaMol benchmark dataset Brown et al. [2019].

All methods are implemented in Python 3.7.6. We used the gSpan library[1] to obtain building blocks. All neural networks and reinforcement learning are implemented in RLlib Liang et al. [2018] and PyTorch Paszke et al. [2019]. All experiments were conducted on Ubuntu 18.04.5LTS with 40 cores of 2.2 GHz Intel Xeon CPU E5-2698 v4, 256GB of memory, and 32GB Nvidia Tesla V100.

Dataset. We use the ZINC 250k molecular dataset, which is a freely available drug-like molecular database [Irwin et al., 2012]. There are 249,456 molecules in total, and the maximum numbers of nodes and edges are 38 and 45, respectively. The number of node labels is nine: {B, C, F, H, I, N, O, P, S}. Molecules are prepossessed by RDKit [Landrum, 2006] so that they are treated as graphs. Before applying MOLDR, we converted molecules in the ZINC dataset into junction trees as we explained in Section 2.3.2. As a result, the number of cliques is 784, which are used as the node label of junction trees. The maximum number of nodes and edges in junction trees become 31 and 30, respectively. The number of molecules on the GuacaMol dataset is 1,591,378 in total, and the maximum numbers of nodes and edges are 88 and 87. The number of node labels is 12: {B, Br, C, Cl, F, I, N, O, P, S, Se, Si}. All molecules are prepossessed by RDKit Landrum [2006] so that they are treated as graphs.

Experiment setting. The gSpan algorithm was applied to converted junction trees with the minimum support of 10,000, 5,000, 1,000, and 100, where we enumerated molecules with more than 7 nodes. To see the effectiveness of our junction tree-based enumeration, we also applied gSpan to the original ZINC dataset without junction tree conversion. In the molecular reassembling step in MOLDR, we use building blocks extracted under the condition of the minimum support 1,000. In MCTS, we set the parameter $c_{puct} = 10$, the terminal condition is when the $\log P$ exceeds 1,000 or QED exceeds 350. In reinforcement learning, we use the policy network with 3-layer MLPs (256, 128, 128 hidden units) to take actions (to choose building blocks and compute state value function), and the activation function is the ReLu function. The generalized advantage estimate (GAE) parameters are set as $\lambda = 1.0$ and $\gamma = 0.99$. The optimizer is stochastic gradient descent, where the mini-batch size is 128 and the learning rate is 5.0×10^{-5}. The terminal condition is when the maximum number of nodes exceeds 100, or the previous reward exceeds the current reward or threshold.

Target properties. As target chemical properties, we employ scores of the *penalized* $\log P$ [Kusner et al., 2017] and Quantitative Estimation of Drug likeness (QED) [Bickerton et al., 2012]. These values are widely used as a benchmark of the task of molecular generation. The penalized $\log P$ is a logarithm of the octanol-water partition coefficient with restrictions on the ring size and synthetic

[1] `https://github.com/betterenvi/gSpan`

accessibility(SA) [Ertl and Schuffenhauer, 2009]. SA score is defined as follows:

$$\text{SA} = \text{Fragment Score} - \text{Complex Penalty}.$$

The Fragment Score was introduced to capture the "historical synthetic knowledge" by analyzing common structural features in a large number of already synthesized molecules. Complex Penalty is computed from summation of each term as follows:

$$\begin{aligned}
\text{Ring complexity} &= \log(\text{nRingBridgeAtoms} + 1) + \log(\text{nSprioAtoms} - 1),\\
\text{Stereo complexity} &= \log(\text{nStereoCenters} + 1),\\
\text{Macro Cycle Penalty} &= \log(\text{nMacroCycles} + 1),\\
\text{Size penalty} &= \text{nAtoms}^{1.005} - \text{nAtoms},
\end{aligned}$$

where "n" denotes the number. We used the penalized log P normalized with the ZINC250k dataset to compare the same setting with other methods, thus direct comparison of scores is fair.

QED is a score representing the drug-like nature of molecular structures. QED represents the function of weighted chemical properties:

$$\text{QED} = \exp\left(\frac{\sum w_i \log d_i}{\sum w}\right),$$

where w is the weight of a molecule and each d_i is one of the following chemical property: molecular weight (MW), octanol-water partition coefficient (ALOGP), number of hydrogen bond donors (HBD), number of hydrogen bond acceptors (HBA), molecular polar surface area (PSA), number of rotatable bonds (ROTB), the number of aromatic rings (AROM), or number of structural alerts (ALERTS). Thus optimizing QED implies to generate the molecules subject to these parameters.

For guiding target values of property, we optimize molecule with $\log P = 8.0$. For multi-objective benchmark, we generate molecules such that QED is higher and SA is smaller (easy to synthesize). We choose the objective function proposed by Tan et al. defined as follows:

$$f(G) = \max\big(\,\text{QED}(G) - 0.1\text{SA}(G)\,\big).$$

Distribution benchmarks. To investigate whether MOLDR can generate diverse molecules or not, we use the GuacaMol benchmark dataset. The proposed scores are listed as follows:

- Validity: whether the generated molecules are actually valid computed in RDKit.
- Uniqueness: the ratio of molecules that are not duplicated to generate molecules.

- Novelty: the ratio of molecules that are not duplicated to original dataset.
- Kullback-Leibler (KD) divergence: measures how well a probability distribution Q approximates another distribution P: $D_{\mathrm{KL}} = \sum_i P(i) \log \frac{P(i)}{Q(i)}$. For the training set and generated set, the probability is calculated from physiochemical descriptors: BertzCT, MolLogP, MolWt, TPSA, NumHAcceptors, NumHDonors, NumRotatableBonds, NumAliphaticRings, and NumAromaticRings by RDKit. The final score is calculated by $S = \frac{1}{k} \sum_i^k \exp(-D_{\mathrm{KL}}, i)$, where k is the number of descriptors (in this case $k = 9$).
- Fréchet ChemNet Distance (FCD). Preuer et al. introduced the Fréchet ChemNet Distance as a measure of how close distributions of generated data are to the distribution of molecules in the training set. Low FCD values characterize similar molecule distributions. Final score is given by $S = \exp(-0.2\mathrm{FCD})$.

Rediscovery molecules. We examined whether or not MOLDR can reconstruct target molecules such as drugs, and can generate molecules exceeding some threshold of similarity between molecules, not just generating molecules with chemical properties. In such cases, we chose Celecoxib, Troglitazone and Thiothixene as rediscovery benchmarks, Aripiprazole as similarity benchmark, and Ranolazine and Osimertinib as MPO benchmarks.

Comparison partners. For comparison, we used scores of $\log P$ and QED reported by [You et al., 2018]. The list of methods are as follows: Junction Tree Variational Autoencoder (JT-VAE) [Jin et al., 2018], Graph Convolutional Policy Network (GCPN) [You et al., 2018], and Objective-Reinforced Generative Adversarial Networks (ORGAN) [Guimaraes et al., 2017]. JT-VAE is a VAE model that generates junction trees of molecules, while GCPN is a method that generates graphs node-by-node using graph neural networks and reinforcement learning to satisfy the objective properties. ORGAN is a model that generates SMILES by Sequence based Generative Adversarial Networks and optimizes the properties through reinforcement learning.

2.4.1 Results of log P and QED

Table 2.1 shows results of applying gSpan to the ZINC database with varying the minimum support. We compare the number of obtained subgraphs and calculation time with or without molecular junction trees. We can see that enumeration based on junction trees is much faster than directly applying gSpan to molecular graphs. This result means that our junction tree-based enumeration is effective in the real-world ZINC database. When the minimum support is 100, we were able to find a large number of subgraphs (23,616 subgraphs), and it is expected that we have collected

Table 2.1: Comparison of frequent subgraph enumeration with or without junction trees.

Minimum support	Number of mined trees	Number of mined graphs
100,000	0	23 (1334 sec)
10,000	8 (164.42 sec)	4040 (106.5 min)
5,000	39 (216.21 sec)	—
1,000	910 (342.20 sec)	—
100	23,616 (775.23 sec)	—

"—" means that computation did not stop in 2 hours

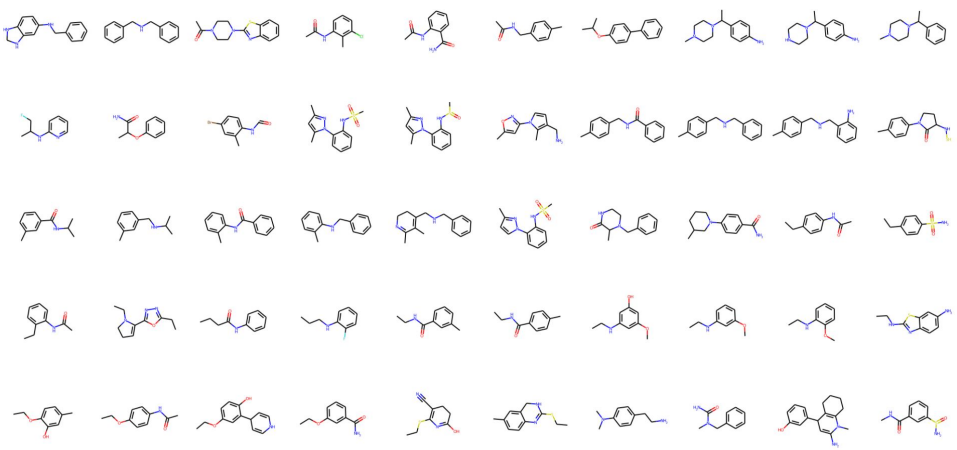


Figure 2.5: Examples of extracted substructures sorted by the score of QED. ZINC 250k molecules are decomposed into junction trees, frequent subtrees are enumerated by gSpan, and they are reconstructed into molecules by ISMAGS. These substructures become building blocks for molecular reassembling.

enough amount of substructures. Therefore we stop decreasing the minimum support. Figure 2.5 shows examples of building blocks of substructures extracted from ZINC 250k filter by the score of QED > 0.7. The obtained substructures are frequent subgraphs with minimum support 100. These structures are used as building blocks for molecular graph reassembling.

Table 2.2 shows top 3 generated molecules according to property scores of the pernalized log P or QED. Scores of other methods come from [You et al., 2018]. MOLDR is similar to the method of JT-VAE as both methods use junction trees, while MOLDR outperforms both scores. The log P is related to lipophilicity (fat solubility) and hydrophilicity (water solubility) of a molecule. Hence, if nodes in a generated molecule have have many carbon (C) and less imide (=NH) or hydroxyl

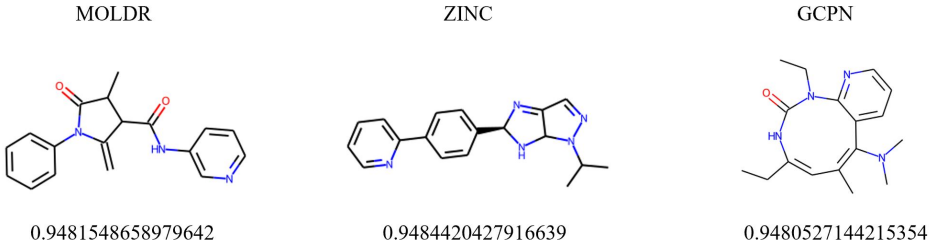


Figure 2.6: Comparison of Top 1 QED molecules in MOLDR, ZINC, GCPN.

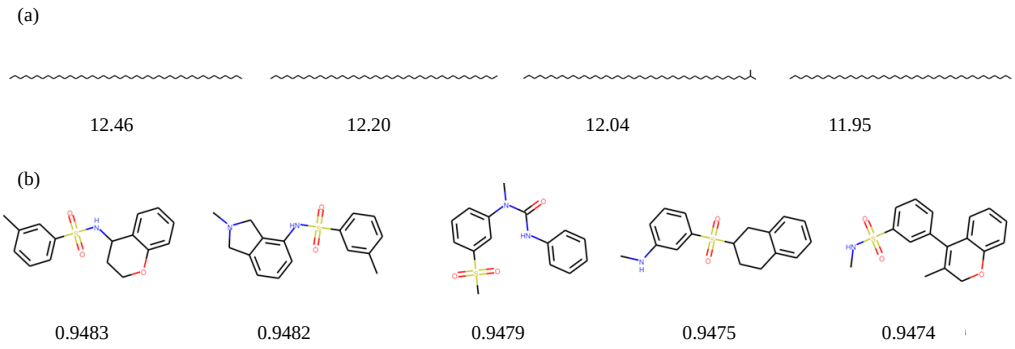


Figure 2.7: Generated molecules based on ZINC dataset by MOLDR+PPO with penalized log P and QED scores.

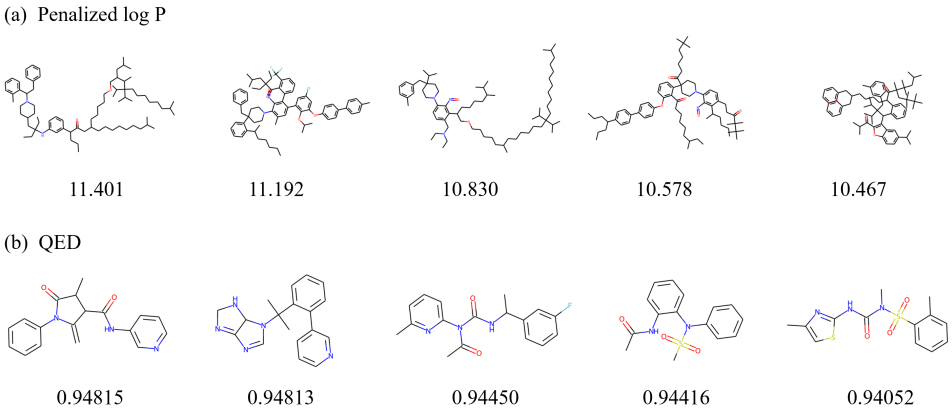


Figure 2.8: Top 5 samples of generated molecules optimized penalized log P and QED respectively by using MCTS.

Table 2.2: Comparison of the top 3 property scores of generated molecules. Scores for ORGAN, JT-VAE, and GCPN are from [You et al., 2018]. The score of penalized log P is normalized in the ZINC dataset.

Method	Penalized logP				QED			
	1st	2nd	3rd	Validity	1st	2nd	3rd	Validity
ZINC	4.52	4.30	4.23	100%	0.948	0.948	0.948	100.0%
ORGAN	3.63	3.49	3.44	0.4%	0.838	0.814	0.814	2.2%
JT-VAE	5.30	4.93	4.49	100.0%	0.925	0.911	0.910	100.0%
GCPN	7.98	7.85	7.80	100.0%	0.948	0.947	**0.946**	100.0%
MOLDR(MCTS)	**11.40**	**11.19**	**10.83**	100.0%	**0.948**	**0.948**	0.944	100.0%
MOLDR(PPO)	**12.46**	**12.20**	**12.04**	100.0%	**0.948**	**0.948**	**0.947**	100.0%

groups (-OH), the resulting log P becomes high. This means that, the larger the number of the atom C is, the higher the log P value is. While we show penalized log P scores in which the ring size and synthetic accessibility are penalized in Table 2.2, if we compute the non-penalized log P score, it tends to be higher and the score is 22.47 for the top molecule of MOLDR. (For the comparison, 1st score in GCPN is 14.49). Moreover, when the penalized log P is not normalized in ZINC 250k, the top score of MOLDR is 17.39, which outperforms the top score 15.13 recently reported in MP-MCTS [Yang et al., 2021] and the score 10.96 of GCPN. In the case of penalized log P optimization, an approach of greedy search such as selecting only C can be enough to maximize the score, because the calculation of the log P score consists of an additive condition. However, it is shown that MOLDR can explore and exploit molecules with optimizing the penalized $logP$ without such trick.

In contrast, the QED score is empirically derived from the combination of various chemical properties and chemical structures. Hence it is not straightforward to maximize QED unlike the case of log P and is more difficult. MOLDR outperforms the score of JT-VAE, and top-1 and -2 molecules generated by GCPN. One can see that our method MOLDR actually outperforms GCPN from more accurate numbers presented in Table 2.5. In order to increase the QED score, generated molecules need to follow the strict restriction of structures. Figure 2.7 and 2.8 illustrate examples of generated molecules with optimization of the penalized log P and QED, respectively, by MOLDR. In penalized log P optimization, the molecular size is larger and molecules include the large number of C. In QED optimization, the size of molecule is smaller than the case of log P optimization, and they have subgraphs that contribute to the QED. We remove the similar graphs that have the highest score in Figure2.9 to show variety of generated molecules by our method.

Table 3 shows the detail of the properties of generated molecules. Theoretically, QED can have a value between 0 and 1, while if we use the default weight w of RDKit for QED computation, the theoretical maximum value of QED is 0.948449.

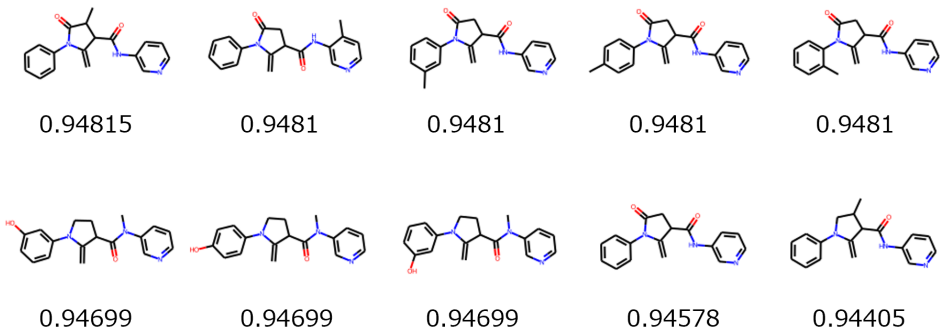


Figure 2.9: Generated molecules by MOLDR+MCTS with their QED scores. The score of QED is high for these molecules because the properties of each molecule lie within the condition in Table 2.5.

Due to our approach of reassembling subgraphs, the second best generated molecule shown in Figure 2.6 has the same components of molecules in the ZINC database with slightly different positions. The molecule with the highest QED score is already included in ZINC, which is the nearest to the optimal QED score. It is interesting that a number of property scores used in QED computation, such as discrete scores like HBA and HDB, are exactly the same with the optimal scores even if each of these scores is not directly optimized in molecular generation. Our method tends to generate similar molecules in QED optimization as shown in Figure 2.9, which may be from the property of graph mining that similar frequent subgraphs are often enumerated such as only one node is different with each other.

2.5 Single-objective optimization

MOLDR is flexible in the sense that it can generate molecules with not only maximizing the target value like $\log P$ but controlling it to be a specific value. As an example, we show molecules generated by MOLDR with specifying the target value $\log P = 8.0$ in Figure 2.10.

2.6 Multi-objective optimization

In a multi-objective task with both QED and SA, Figure 2.11 shows results of generated molecules when only QED is optimized or both QED and SA are optimized. If only QED is optimized (Figure 2.11(a), (b)), generated molecules tend to have a higher QED score with SA score being around 3 to 5. Otherwise if

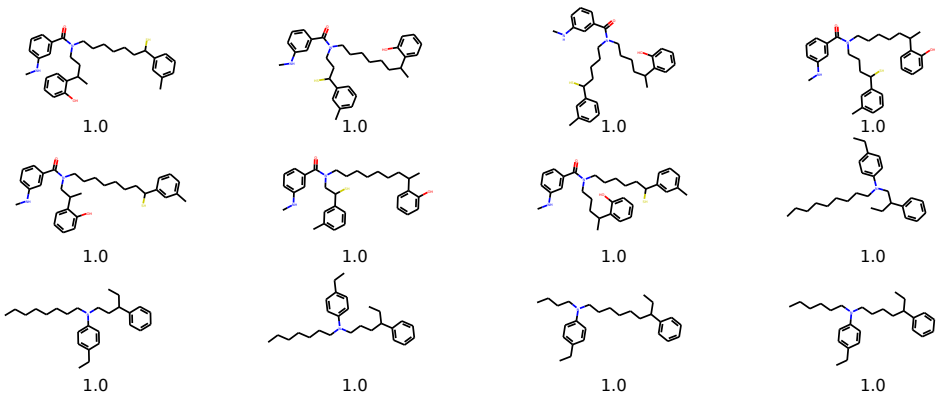
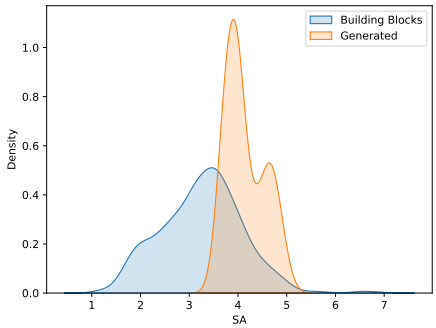
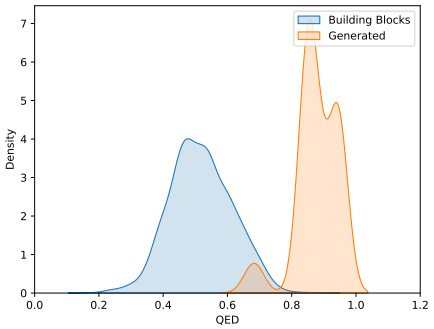


Figure 2.10: Generated molecules with log P = 8.0

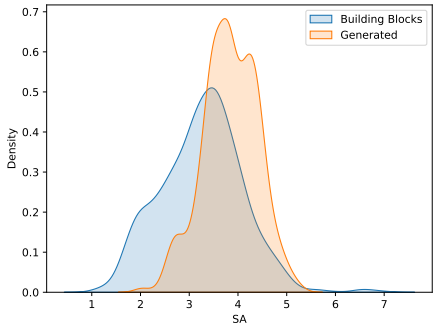
both QED and SA are optimized, the distribution of SA shifts left (Figure 2.11(c), (d)) compared to the case of QED optimization.



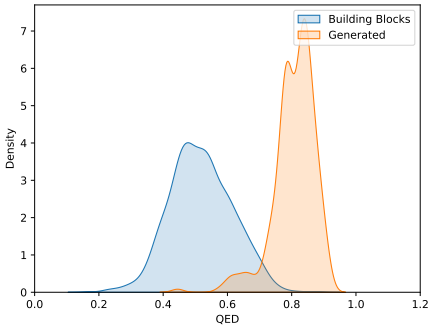
(a) SA distribution of building blocks and generated molecules when QED is optimized.



(b) QED distribution of building blocks and generated molecules when QED is optimized.



(c) SA distribution of building blocks and generated molecules when both QED and SA are optimized simultaneously.



(d) QED distribution of building blocks and generated molecules when both QED and SA are optimized simultaneously.

Figure 2.11: Distribution of generated compounds for optimizing both QED and SA in the multi-objective task on the GuacaMol dataset.

Table 2.3: Distribution benchmarks at 10k molecules on GuacaMol dataset.

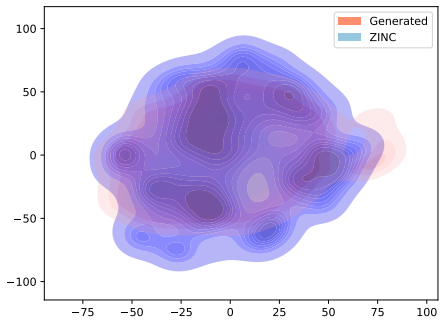
benchmark	Random Sampler	Graph MCTS	SMILES LSTM	VAE	**MOLDR** (Random)
Validity	1.00	**1.000**	0.959	0.870	**1.000**
Uniqueness	0.997	**1.000**	**1.000**	0.999	0.994
Novelty	0.000	0.994	0.912	0.974	**0.996**
KL divergence	0.998	0.522	**0.991**	0.982	0.442
Fréchet ChemNet Distance	0.929	0.015	**0.913**	0.863	0.029

2.6.1 Results of GuacaMol

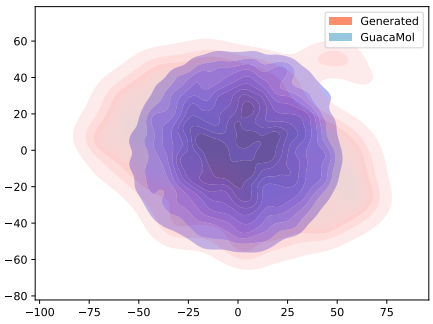
GuacaMol benchmark evaluates whether or not a model can generate valid, unique, and novel molecules from a training dataset. The KL (Kullback–Leibler) divergence and the Fréchet ChemNet distance (FCD) between a training set and generated molecules are also used. In a decomposition step, 1,709 building blocks are mined from the GuacaMol dataset with minsup = $10,000$. The distribution benchmark is evaluated from 10k sampled molecules in a reassembling step. MOLDR can generate valid molecules due to reassembling a building block of molecules. Although the uniqueness is slightly smaller than other models, MOLDR depends on random seeds to choose building blocks. In terms of the KL divergence and the FCD, MOLDR is inferior to SMILES LSTM and VAE. However, the score is similar to Graph MCTS because it is also a similar strategy to generate molecules. In addition, MOLDR can sample molecules randomly from an untrained policy network. Hence, MOLDR has a potential to generated molecules that are largely different from those in the training dataset, leading to lower scores of the KL divergence and the FCD. In order to improve the performance in terms of the KL divergence and the FCD, MOLDR would need to train the policy network and design an appropriate reward function, such as the similarity between the training dataset and generated molecules.

Figure 2.12 shows results of the distribution of generated molecules when trained on the ZINC or the GuacaMol dataset. Molecules are first mapped into 300-dimensional vectors using Mol2Vec [Jaeger et al., 2018] and t-distributed stochastic neighbor embedding (t-SNE) is applied to visualize the distribution of generated molecules and that of the training set. On the ZINC dataset, generated molecules are mostly overlapped within the training set, while on the GuacaMol dataset, the distribution of generated molecules goes beyond that of the training set. It means that the generated molecules are not similar to the training set; therefore, the KL divergence and FCD scores are likely to be lower. However, random sampling from MOLDR can generate larger molecules out of distribution.

Table 2.4 shows the result of rediscovery benchmarks. MOLDR can generate the target molecules with high accuracy, whose scores are competitive with SMILES



(a) Distribution on ZINC dataset



(b) Distribution on GuacaMol dataset

Figure 2.12: Distributions of training sets in GuacaMol and ZINC dataset, and generated molecules. Molecules are mapped into vectors using Mol2vec, and then we apply t-SNE dimensionality reduction for visualization.

LSTM and Graph GA. The most notable difference between those models is that our model can visualize the generating process of molecules, not just generating the target molecule. Although in SMILES LSTM, intermediate molecules are evaluated from the state value function, and the SMILES character is selected based on the state, it is not easy to interpret why the character is vital at a particular time step, especially when generating a ring. In contrast, in MOLDR, building blocks are directly selected, and the sub-structure affects the target directly. Generating process is shown in Figure 2.14 and 2.13. Since the generation performance of MOLDR depends on the building blocks obtained from graph mining, in practical applications, it is important to prepare an appropriate dataset and set an appropriate minimum support based on a priori knowledge.

Table 2.4: Goal Directed benchmarks

Benchmark	Best in dataset	SMILES LSTM	Graph GA	MOLDR
Celecoxib rediscovery	0.505	1.000	1.000	1.000
Troglitazone rediscovery	0.419	1.000	1.000	1.000
Aripiprazole similarity	0.595	1.000	1.000	1.000
Osimertinib MPO	0.839	0.907	0.953	0.898
Ranolazine MPO	0.792	0.855	0.920	0.864

Limitation. Our method MOLDR cannot design a molecule that has the large cyclic structure if these structures are not included in a database as necessary building blocks to construct such structure are not detected by frequent subgraph

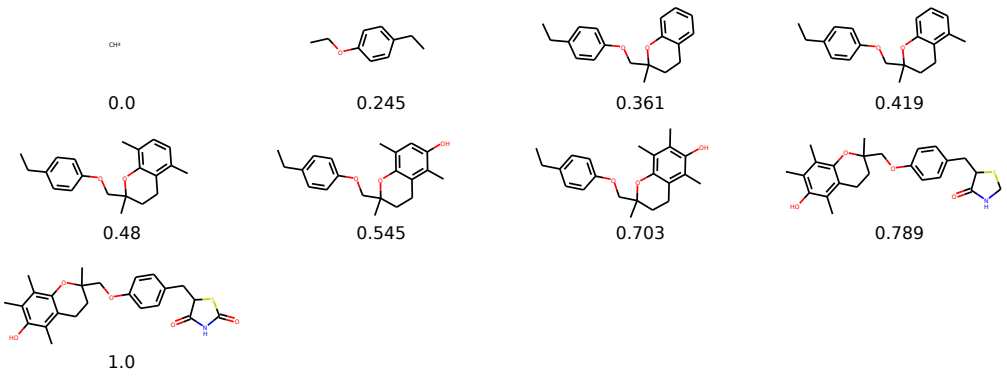


Figure 2.13: Generating process of Troglitazone rediscovery. The number under the molecules denotes the similarity score between a generated molecule and target. MOLDR can generate Troglitazone in 8 steps.

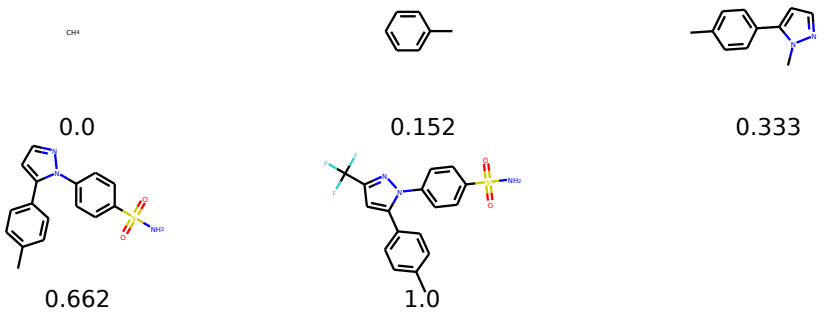


Figure 2.14: Generating process on Celecoxib rediscovery.

mining. In addition, as we state in Section 3.3, the bigger the size of a graph is, the higher the number of candidates to merge nodes and edges is, resulting in too high computational cost.

Table 2.5: Details of top molecules in terms of QED.

Method	QED	MW	ALOGP	HBA	HBD	PSA	ROTB	AROM	ALERTS
Optimized	0.948449	306.051	2.8134	3	1	53.05	3	2	0
ZINC	0.948442	305.385	2.8273	3	1	52.88	3	2	0
MOLDR	0.948155	307.353	2.8329	3	1	62.30	3	2	0
GCPN	0.948052	300.406	2.8057	3	1	53.92	3	2	0

2.7 Conclusion

We have proposed a new molecular generation method, called MOLDR, which decomposes graph structures and reassembles them. In our experiments on the ZINC database, MOLDR can find a better molecules in terms of two properties, the penalized log P and the drug-likenss score QED, than the state-of-the-art molecular generation methods. Our approach is general, hence it can be also applied to any graph generation problem as well as molecular graph generation.

For our future work, it is interesting to explore clustering of subgraphs extracted by graph mining as there are often many graphs that are similar with each other. Since graph construction step is interpretable in MOLDR, incorporating MOLDR with the retrosynthesis analysis to consider the chemical reaction is an interesting topic.

Chapter 3

Substructure-based Machine Learning

3.1 Introduction

Graph-structured data are ubiquitous in various domains from social networks to system engineering and bio- and chemo-informatics. Most of the recent machine learning methods for graphs are essentially based on the *message passing scheme*, which aggregates feature vectors associated with neighboring nodes and updates them via a certain function at each node [Hamilton et al., 2017, Xu et al., 2019, Veličković et al., 2018, Gilmer et al., 2017, Bo et al., 2021]. This scheme is widely employed to learn the representation of nodes and graphs in recently proposed *graph neural networks* (GNNs).

In machine learning on graphs, *graph kernels* have been studied to measure the similarity between graphs, which can be combined with classification or prediction tasks to achieve good predictive performances on various graph datasets [Borgwardt et al., 2020, Kashima et al., 2003, Sugiyama and Borgwardt, 2015]. If node features in graphs are discrete – that is, each node has its label – the Weisfeiler–Lehman (WL) sub-tree graph kernel can be the first choice for graph classification tasks, which also shares the message passing algorithm in its process [Shervashidze et al., 2011b]. This "WL scheme" can effectively and efficiently count subgraph patterns in graphs, resulting in the high predictive performance in ML tasks for graphs. The WL scheme obtains feature vectors based on counting the node label appearance on neighboring nodes, which implicitly obtains representation of graphs and is likely to represent some specific characteristics that are able to explain targets in graph ML tasks. Although huge computational cost is a limitation of graph kernel approaches, the quadratic time complexity with respect to the number of graphs in construction of a kernel matrix, when applied to larger datasets, make their performance superior

to other approaches in various medium sized datasets [Borgwardt et al., 2020].

GNNs have emerged as a way of learning representation on graphs. The message passing scheme is commonly used in most of proposed GNNs [Gilmer et al., 2017, Kipf and Welling, 2017, Xu et al., 2019, Scarselli et al., 2009]. For example, graph convolutional network (GCN) is known to be a simple GNN model for semi-supervised learning [Kipf and Welling, 2017], which is similar to the WL scheme. Graph isomorphism network (GIN) is more similar to the WL graph kernel, which have been theoretically and empirically shown to be as powerful as the WL scheme with respect to their discriminability and representation power [Xu et al., 2019]. In addition, to learn better representation of node features and overcome the limitation of graph convolutions, attention-based graph neural models have been studied [Veličković et al., 2018, Brody et al., 2021]. These graph neural network models are helpful to solve graph prediction tasks as well as node or link prediction.

In GNNs, the problem of over-smoothing, where the similarity between nodes becomes closer and closer when more and more message passing is performed, is one of common issues. The over-smoothing deteriorates the quality of graph representation, particularly when multiple convolutional layers are stacked, because of the low information-to-noise ratio of the message received by the nodes, which is partially determined by the graph topology [Chen et al., 2020]. As a result, the performance of a trained model such as the accuracy of classification worsens when over-smoothing occurs. This means that the message passing scheme may inherently have a limitation in terms of capturing large subgraph structures as it requires many message passing iterations. Thus, if such large subgraphs are relevant in prediction, this would be a serious problem for GNNs.

In this chapter, to investigate the advantage and the limitation of the message passing scheme in GNNs and graph kernels, we focus on the influence of the over-smoothing problem. More precisely, we empirically examine how features of graph substructures obtained by the WL (or message passing) scheme affect the predictive performance as the number of convolutional layers increases. First, we compare the WL sub-tree graph kernels with GNNs, including GCNs and GINs on classification and regression real-world graph datasets collected from a variety of domains. In GNNs, for certain types of datasets, we show that the large number of message passing iterations deteriorates the predictive performance due to the over-smoothing phenomenon. Second, we argue that the averaged feature vectors over nodes learned from graph convolutional layers, which are expected to represent the overall graph characteristics, are not stable as the number of message passing iterations increases. Finally, we evaluate graph features on multi-layer perceptrons (MLPs) and support vector machines (SVMs) in the task of classification and regression. We also show that SVMs with features obtained from GCNs can be used as a way of measuring how strong the contribution of node features is, where the node features are concatenated at each message passing iteration.

Our contributions are summarized as follows:

- The prediction performance of the WL kernel outperforms the most fundamental GNNs, GCNs and GINs, if the number of message passing iteration increases.
- The WL kernel does not deteriorate significantly for many message passing iterations in most of datasets, while GCNs and GINs do deteriorate due to their large number of parameters and ill-trained previous node features.
- Transition of node features tend to be small, which leads to difficulty of determining which feature is informative for prediction, and the features converge to indistinguishable values when the training fails.

3.2 Methods for Graph Machine Learning

3.2.1 Notation

A *graph* is a tuple $G = (V, E)$, where V and E denote the set of nodes and edges, respectively. Nodes and edges can have labels (attributes) via label functions $l_V : V \to \Sigma_V$ for nodes and $l_E : E \to \Sigma_E$ for edges with some label domain Σ_V and Σ_E, which can be any set such as $\mathbb{Z}$ and $\mathbb{R}^d$.

For two graphs $G = (V, E)$ and $G' = (V', E')$, we say that G' is a *subgraph* of G, denoted by $G' \sqsubseteq G$, if $V' \subseteq V$ and $E' \subseteq (V' \times V') \cap E$.

3.2.2 The Vertex Histogram Kernel

One of the simplest graph kernels is the vertex histogram kernel, which counts node label matching between a pair of graphs. Let d be the number of types of node labels – that is, $d = |\Sigma_v|$ – and assume that $\Sigma_v = \{1, 2, \ldots, d\}$ without loss of generality. The histogram vector $\mathbf{f}_i = (f_{i,1}, \ldots, f_{i,d})$ of node labels of a graph G_i is given as $f_{i,m} = |\{v \in V : l_v = m\}|$ for each $m \in \Sigma_v$. The vertex histogram kernel k_{VH} between G and G' is defined as

$$k_{\mathrm{VH}}(G_i, G_j) = \langle \mathbf{f}_i, \mathbf{f}_j \rangle. \tag{3.1}$$

3.2.3 The Weisfeiler–Lehman Kernel

The Weisfeiler–Lehman (WL) sub-tree kernel is based on the Weisfeiler–Lehman test of isomorphism. The key idea of the kernel is the "WL scheme", which is re-labeling of node labels by gathering neighboring node labels. Then each new

label is a good proxy of an indicator of subgraph structure. This re-labeling step is repeated until every pair of graphs have different node labels or reaching at a certain step t, which is a hyperparameter. If node label sets are different for two graphs, this indicates that they are not isomorphic with each other. Moreover, node labels obtained from this procedure implies its subgraph structures, resulting in the tree structure of labels. After the procedure, each graph can be represented by the occurrence of labels (subgraph structures).

In the WL kernels, these subgraph structures obtained from the WL scheme are mapped into the Hilbert space. In other words, the obtained node label sets are used to compute the similarity between graphs. The WL kernel is also based on the vertex histogram, but it extends the original vertex histogram through iterations of node re-labeling. The final kernel matrix after t iteration of the WL scheme is given as follows:

$$k_{WL}(G_i, G_j) = k_{\mathrm{VH}}(G_i^{(0)}, G_j^{(0)}) + k_{\mathrm{VH}}(G_i^{(1)}, G_j^{(1)}) + \cdots + k_{\mathrm{VH}}(G_i^{(t)}, G_j^{(t)}), \quad (3.2)$$

where $G^{(t)}$ denotes the graph G with the updated node labels obtained by tth iteration of the WL scheme. In the WL scheme, each node label $l_v^{(t)}$ for a node $v \in V$ of $G = (V, E)$ at tth iteration is relabeled as $l_v^{(t+1)}$ by the hash function given as

$$l_v^{(t+1)} = \mathrm{HASH}\Big(\mathrm{CONCAT}\left(l_v^{(t)}, \mathcal{N}(v)\right)\Big), \quad l_v^{(0)} = l_v, \quad (3.3)$$

where $\mathcal{N}(v)$ is the set of neighboring node labels of v. The function HASH denotes the aggregated labels into unique new labels, and CONCAT denotes the operation of concatenating labels. As the number t of iterations increases, the kernel can capture more and more overall information of graph structure. When t goes to infinity, it always converges to some value.

3.2.4 Graphlet kernel

The graphlet kernel is calculated by the frequency of occurrence of matched subgraphs, called graphlets [Pržulj, 2007]. Graphs are decomposed into graphlets. Let $\mathcal{G} = \{\mathrm{graphlet}_1, \mathrm{graphlet}_2, \ldots, \mathrm{graphlet}_{|\mathcal{G}|}\}$ be the set of size-k graphlets in G. Using the histogram vector $\mathbf{f}_i = (f_{i,1}, \ldots, f_{i,|\mathcal{G}|})$ such that $f_{i,m} = \#(\mathrm{graphlet}_m \sqsubseteq G)$ [Shervashidze et al., 2009]. Then the graphlet kernel is defined as

$$k(G_i, G_j) = \langle \mathbf{f}_i, \mathbf{f}_j \rangle. \quad (3.4)$$

3.2.5 Graph convolutional neural networks

Graph convolutional neural networks are first introduced for semi-supervised learning of classifying nodes in a graph. In graph convolutional networks, node

labels are treated as a feature matrix $H \in \mathbb{R}^{|V|\times d}$, where each row is the corresponding node label. The update rule of H from $H^{(t)}$ to $H^{(t+1)}$ at the tth iteration in a graph convolutional network is defined as

$$H^{(t+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{\frac{1}{2}}H^{(t)}W^{(t)}), \tag{3.5}$$

where $\tilde{A} = A + I$ is the adjacency matrix of G with self-loop (I is the identity matrix), $\tilde{D}_{ij} = \sum_j \tilde{A}$ is the degree matrix with self-loop, $W^{(t)}$ is a matrix composed of trainable weight parameters in a layer t used for mapping of node feature matrix H, and σ is a non-linear function such as $\mathrm{ReLu}(\cdot) = \max(0, \cdot)$. The initial node feature matrix $H^{(0)} \in \mathbb{R}^{|V|\times d}$ is directly given by node labels of a given graph G. This approach is related to the continuous version of the WL scheme with an additional mapping and non-linear transformation instead of concatenating node features [Kipf and Welling, 2017].

For regression and classification settings, the output of node features at the final layer is reduced by the pooling operation, such as taking the mean and the sum over vertices, which is equivalent to the readout operation that can obtain the representation of graph features. The readout graph feature is passed into MLP layers and trained through back propagation.

In computing for classification and regression, the reduce function that returns the final output value o_G for a given graph G is defined as simply sum over nodes

$$o_G = \mathrm{MLP}(\mathrm{READOUT}_{\mathrm{GCN}}(H^{(T)})), \tag{3.6}$$

where $\mathrm{READOUT}_{\mathrm{GCN}}(\cdot)$ returns the sum pooling, the sum of each column, of an input feature matrix. Since each row of H corresponds to each node of a graph G, the $\mathrm{READOUT}_{\mathrm{GCN}}$ operation performs the summation of feature vectors across nodes.

3.2.6 Graph Isomorphism Networks

The graph isomorphism network (GIN) is a model that is known to be as powerful as the WL test [Xu et al., 2019]. To update the representation of each node feature vector h, the GIN employs MLP as the core process in the following manner.

$$h_v^{(t+1)} = \mathrm{MLP}^{(t)}\left((1+\epsilon^{(t)})h_v^{(t)} + \sum_{u\in\mathcal{N}(v)} h_u^{(t)}\right), \tag{3.7}$$

where $\mathrm{MLP}^{(t)}$ is a MLP layer at tth iteration, $\epsilon^{(t)}$ is a learnable weight or a fixed scalar at the tth iteration. For graph classification and regression tasks, the output

of graph features $h_G^{(T)}$ at the Tth iteration is defined as follows:

$$h_G^{(T)} = \mathrm{CONCAT}\left(\left(\mathrm{READOUT}\left((h_v^{(t)} \mid v \in G)\right) \mid t = 0, 1, \ldots, T\right)\right). \tag{3.8}$$

The READOUT function aggregates node features from the final iteration to obtain graph features of $h_G^{(t)}$ by the summation or the graph-level pooling function. In each iteration, graph features are read out and concatenated as total features. The CONCAT function is defined as

$$h_G = \sum_{t=0}^{T} w^{(t)} h_G^{(t)}, \tag{3.9}$$

where w is the weights of linear layers. The final features of h_G are computed through each linear layer.

3.2.7 Special case of GIN (Pre-fixed GCN)

Our focus in this chapter is not on developing state-of-the-art GNNs, but on examining the effectiveness of GNNs with respect to treating large substructures of graphs. Therefore, we fix every weight to be the identity matrix and ϵ to be zero in GINs. In such a situation, the update of node features is fully based on graph structure (adjacency matrix) itself. The formula is defined as follows:

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{\frac{1}{2}}H^{(l)}), \tag{3.10}$$

$$H_G = \mathrm{CONCAT}\left(\mathrm{READOUT}(H_i^{(t)}) | t = 0, 1, \ldots, T\right). \tag{3.11}$$

Note that the obtained features can be combined with not only MLPs but also any machine learning models. In SVMs, the graph feature of H_G is decomposed into a collection of graph features obtained via iterations and the resulting kernel is obtained like the WL kernel as

$$k(H_G, \cdot) = k(H_G^{(0)}, \cdot) + k(H_G^{(1)}, \cdot) + \cdots + k(H_G^{(T)}, \cdot). \tag{3.12}$$

Since the dimensionality of concatenated features of graphs becomes larger and larger if more and more iterations are performed, we introduce the norm to normalize each feature matrix as follows:

$$H^{(t+1)} = \frac{\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{\frac{1}{2}}H^{(t)}}{|\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{\frac{1}{2}}H^{(t)}|}. \tag{3.13}$$

We call these features pre-fixed GCN as the weights of GCN is prefixed into unity.

3.2.8 Measuring Transition of Node Features

To measure the degree of transition of node features after weight training in graph convolutional layers, we simply introduce the AVERAGE operation of node feature vectors for each convolutional layer. In GCNs, learned node features of H are updated per layer so that we retrieve and average $H_G^{(t)}$ over nodes at each iteration step t, and we can describe the change of node features with the size f, which is the number of node features.

$$H_G^{(t)} = \begin{pmatrix} h_{11}^{(t)} & \cdots & h_{1f}^{(t)} \\ \vdots & \ddots & \vdots \\ h_{v1}^{(t)} & \cdots & h_{vf}^{(t)} \end{pmatrix}. \tag{3.14}$$

The update of H is done with Eq. (3.5). The dimensionality of these features is reduced by the mean pooling as

$$h_G^{(t)} = \left(\frac{1}{|V_G|} \sum_v^{|V_G|} h_{v1}^{(t)}, \frac{1}{|V_G|} \sum_v^{|V_G|} h_{v2}^{(t)}, \ldots, \frac{1}{|V_G|} \sum_v^{|V_G|} h_{vf}^{(t)} \right), \tag{3.15}$$

$$h_G^{(t+1)} = \left(\frac{1}{|V_G|} \sum_v^{|V_G|} h_{v1}^{(t+1)}, \frac{1}{|V_G|} \sum_v^{|V_G|} h_{v2}^{(t+1)}, \ldots, \frac{1}{|V_G|} \sum_v^{|V_G|} h_{vf}^{(t+1)} \right). \tag{3.16}$$

We expect that embedded node features include information about the neighboring node features.

3.3 Experiments

3.3.1 Experimental Setting

Datasets

To investigate how sub-graph structures affect the performance in classification and regression tasks, we collected eight classification datasets and three regression datasets from various domains widely used as benchmarks of graph machine learning. In terms of classification datasets, Kersting et al. [Kersting et al., 2016] listed various benchmark sets for graph kernels. Classification benchmarks include five bio-informatics datasets (MUTAG, PTC, NCI1, PROTEINS, and DD) and two social network datasets (IMDB-BINARY and REDDIT-BINARY). Regression benchmarks are three chemo-informatics datasets: ESOL from MoleculeNet for the prediction of water solubility, FreeSolv from MoleculeNet for prediction of hydration free energy of small molecules in water, and Lipophilicity from

MoleculeNet for the prediction of octanol/water distribution coefficient (logD at pH 7.4) ["Wu et al., "2018"]. Node labels in graphs are converted into a one-hot matrix for GNNs while graph kernels use node labels directly to compute gram matrices. As for social network graphs, all node feature vectors are the same as each other according to the procedure proposed in literature [Xu et al., 2019, Siglidis et al., 2020]. Our goal is to investigate whether or not graph machine learning models can appropriately learn information of large subgraph structures when the amount of message passing increases. We do not use edge features and node features obtained from 3D structures such as the distance between nodes for chemo-informatics dataset as they are not relevant to our task. All datasets are available from DGL [Wang et al., 2019] and DGL-LifeSci [Li et al., 2021]. The statistics of datasets are summarized in Table 3.1.

Models and Configuration

We chose two graph neural networks models that are highly related to the WL scheme: Graph Convolutional Network (GCN) [Kipf and Welling, 2017] and Graph Isomorphism Network (GIN) [Xu et al., 2019]. All neural networks are implemented in Deep Graph Library (DGL) [Wang et al., 2019] and PyTorch [Paszke et al., 2019]. The graph kernel of vertex histograms and the WL sub-tree kernel are implemented in GraKel [Siglidis et al., 2020] and GRAPHLETs are obtained using gSpan [Xifeng Yan, 2002] library[1] for subgraph mining. Note that we enumerated all the subgraphs in GRAPHLETs instead of performing sampling. The maximum number of nodes and the support in gSpan is set to 5 and 500 in DD and REDDIT, 5 and 100 in IMDB, and 8 and 1 in other datasets. Support Vector Machines (SVMs) is trained with scikit-learn [Pedregosa et al., 2011]. All methods are implemented in Python 3.7.6. All experiments were conducted on Ubuntu 18.04.5LTS with a single core of 2.2 GHz Intel Xeon CPU E5-2698 v4, 256GB of memory and 32GB Nvidia Tesla V100.

Evaluation

In classification, datasets are evaluated in stratified 10-fold cross-validation, where each fold preserves the class ratio of samples. In regression, we evaluated the performance as root mean squared errors (RMSE) with a holdout validation set, where a training set is 70 percent in the entire dataset. To perform fair comparison of learned node features, the dimensionality of initial weights in convolutional layers is the same as that of initial node features, which is $|V| \times F$ with the number of nodes $|V|$ and that of features F. In classification datasets, the maximum number of trainable parameters in GCN is 234,960 on the DD dataset when 30 graph

[1] `https://github.com/betterenvi/gSpan`

convolutional layers are stacked (the number of initial features (88) × the number of initial features + bias (89) × the number of GCN layers (30)). In regression datasets, the number of trainable parameters in GCN is 166,500 when 30 graph convolutional layers are stacked. The dimensionality of the convolutional weights is also set to be 64 in order to obtain the best accuracy and the RMSE. For both classification and regression settings, after the process for node embedding from GCN layers, the obtained features of graphs are aggregated to pass them to MLP layers with batch normalization. In GNNs, graph datasets are mini-batched according to datasets. The default size is 32. About hyperparameters of networks, training epochs are 350 and two layers of MLPs and its linear hidden dimension is 1024. Adam [Kingma and Ba, 2014] is used as an optimizer and its learning rate is 0.01. Other parameters, such as betas for computing running averages of gradient and its square are default values on PyTorch. The learning rate scheduler is a step function and decays the learning rate half per 50 epochs. The loss function is cross-entropy loss in classification and mean squared loss in regression.

3.3.2 Results and discussion

Classification

In classification, we use the Vertex Histogram kernel as a baseline, which computes the similarity (kernel value) between graphs by simply counting the matching node labels; in so doing, it completely ignores the graph topological information. We also use the GRAPHLET kernel, which includes information of subgraphs under the designated number of vertices.

Classification performance of each method. We show classification accuracy on the bio-informatics and social network datasets in Table 3.2 and Table 3.3. Table 3.2 shows the best accuracy on each dataset when changing the number of iterations of message passing, where hyper-parameters are tuned by the 10-fold cross-validation. In the table, the method "V" at the first row means the vertex histogram kernel that shows our baseline of classification. The kernel shows that most of the datasets can be classified to a certain extent by only counting node labels in graphs. Since the vertex histogram kernel does not include any information about graph topological structures, if the classification accuracy is comparable at a certain level, this means that the effect of subgraphs is marginal.

The WL kernel can consider additional information of subgraph structures through the WL scheme of iteration. The GRAPHLET kernel has a potential to include more information about substructures than the WL kernel according to the setting of maximum number of vertices through enumeration of subgraphs (graphlets). Note that the complexity of enumeration of subgraphs for the GRAPHLET kernel exponentially increases as the number of vertex increases [Shervashidze et al.,

2009], so it is not feasible to set the large number of vertices.

The pre-fixed GCN SVM is the support vector machine with input features computed from GCN layers where the weight is the identity matrix. Such graph features represent the graph structure itself. GCN is related to the WL kernel, while features obtained through graph convolution can include information of subgraph structures as embeddings through training at the final layer. In contrast, GIN can include all features of substructures at each iteration.

In Table 3.2, the accuracy score of the GRAPHLET kernel, which explicitly considers all the possible subgraphs under the certain number of vertices, is highest on the MUTAG and DD dataset. The accuracy of the WL kernel is also high overall and shows the best score in three datasets (PTC-MR, NCI1, and PROTEINS). The accuracy scores of both GIN and GCN are comparable to those of the WL graph kernel. When we consider downstream classification methods in GCNs, SVM with the features computed from GCNs with pre-fixed weights, which can be viewed as the continuous version of the WL scheme and weights in GCNs are not trained and node features depend on the initial node attributes and its structure (the adjacency matrix) can be effective in certain types of datasets as it shows comparable or better accuracy in, for example, MUTAG and DD than the standard GCN. While our result is different from that in [Xu et al., 2019] due to hyperparameter settings, but our comparison is fair as hyperparameters are tuned in the same way across comparison partners.

Effects of the number of message passing iteration. Figures 3.1, 3.2, and 3.3 are classification results of GCNs and GINs when the amount of message passing increases, which investigates the impact of larger subgraph structures on its predictive performance. As the number of iterations increases, GCNs fail to appropriately learn representation of graphs in NCI1 and REDDIT-BINARY, and GINs fail in NCI1, PROTEINS, DD, REDDIT-BINARY, and IMDB-BINARY. One reason for this phenomenon is that GINs have more parameters than GCNs, so it is more difficult to effectively learn representation of node features as it is more affected by the previous ill-learned node features. However, as Tables 3.2 and 3.3 show, GINs are more powerful at learning representation of graphs on various datasets and also comparable to the WL kernel when such hyperparameters are properly tuned.

In the WL kernel, it is easy to compute the kernel even if the number of message passing iterations increases. Figure 3.4 shows the classification accuracy as the number of iterations increases. We have increased the number of iterations in the WL kernel up to 100. In the MUTAG dataset, labels obtained from the WL scheme become unique when iterations reach to 13, which means no more information gains. Small fluctuations of the resulting accuracy comes from the loop structure in graphs. Overall, the accuracy score is not largely affected by the number of iterations in the WL kernel, except for NCI1 and DD. One possible

reason for such deteriorating accuracy in NCI1 and DD is that they have more node labels than other datasets and it becomes difficult to discriminate graphs as the number of iteration increases. Various node labels contribute to the occurrence of multiple types of unique subgraphs. Since graph kernels fundamentally measure the similarity between graphs based on subgraph matching, the kernel value gets smaller if there are many unique subgraphs. In the case of NCI1 and DD, when the number of iterations increases, more and more unique subgraphs appear, and the additional contribution to kernel values becomes smaller and smaller, resulting in the difficulty of discrimination. An interesting observation is that the accuracy on the REDDIT-BINARY dataset increases when the number of iterations increases, although the highest accuracy is at the first iteration.

As for GCNs and GINs, Figures 3.6 and 3.7 show the training-validation loss and accuracy via 10-fold cross-validation per epoch on the MUTAG and NCI1 datasets. In each figure, left and right plots represent the training log of GCNs under the setting of the number of iterations 1 or 30, respectively. When the number of graph convolutional layers is one, the loss and accuracy of training-validation sets are rather smooth compared to the case where the number of graph convolutional layers is 30. After training after 350 epochs, the standard deviation of the training-validation loss and accuracy fluctuates greatly, although the final accuracy score is almost the same, at around 70 percent. This shows the difficulty of learning for a larger number of iterations where there are more trainable parameters as well. Figure 3.3 shows results on the social networks datasets. In GCNs, the accuracy does not change so much compared to GINs. All graphs on social network datasets have the same node labels, and GCNs may be easier to learn representation of node features, while GINs can get the best accuracy at the first or second steps.

Transitions of node features over message passing iteration. To investigate how the number of message passing iterations affects the learning and how node features change through iteration, we plot the transition of node features shown in Figures 3.8 and 3.9. Figure 3.9 shows that node features in GCNs is likely to converge to a certain value on the NCI1 dataset, where over-smoothing is observed. Here, the size of features may be relevant to this phenomenon; that is, when the number of trainable parameters increases in graph convolutional layers, it is more difficult to learn the node features appropriately. As we can see in the MUTAG dataset in Figure 3.8, when the weights are properly trained, even though the transition of node features tends to become small, the classification performance does not change.

Regression

Regression performance of each method. In addition to the classification task, we investigated the regression performance of WL kernels and GCNs. Results

are shown in Table 3.4. In the table, 0-SVR is the linear kernel computed from the sum of initial node features across nodes, which is similar to the Vertex Histogram kernel. In chemo informatics datasets, graph features are often used as fingerprints (subgraphs in molecules) such as extended-connectivity fingerprints (ECFPs) [Rogers and Hahn, 2010b] to deal with larger datasets. We chose smaller datasets for regression because ECFPs have a problem of bit collision, where fingerprints have duplication of subgraph structure in vectors. In the WL kernel, atomic numbers are directly used as node labels. The WL kernel shows the best result (the smallest RMSE) in ESOL and lipophilicity. GCNs and GINs are inferior to the WL kernel for the two datasets, while GCNs show the best score in FreeSolv using node features from two convolutional layers with their training.

Effects of the number of message passing iteration. To investigate the impact of message passing iteration in the regression task, we plot RMSE results of SVR with node features learned from GCNs in Figure 3.10. In the plots, the x-axis (iteration) means the number of message passing (graph convectional layers). In the left plots, the output of node features from each GCNs layer is used in SVR, and in the right plots, concatenated node features from GCNs are used in SVR. In ESOL and Lipophilicyty, an increase in the amount of message passing does not contribute to its predictive performance, while in FreeSolv it makes some contribution to the predictive performance. Concatenating graph features from each layers can be an effective choice when using a large number of iterations. In addition, the place where features from each GCNs layer become a plateau can provide good features. In FreeSolv, the score of GCNs with SVR fluctuates, and concatenating graph features gets the best score, at around five iterations. However, SVR is still effective and, with a little hyperparameter tuning, can learn the model with high predictive performance. The result of the RMSE of GCNs shows that increasing the number of message passing does not contribute to its predictive performance. Especially in GINs, the performance deteriorates, or learning is too unstable to train the model properly. This phenomenon often occurs, as shown in the classification section. However, as can be seen in Fig. 10, node features obtained from each layer in trained GCNs seem to be informative for SVR prediction on ESOL and lipophilicity datasets when concatenating these features. In contrast, the resulting RMSE of FreeSolv shows that calculated node features are likely to include noise because the performance of SVR decreases when they are concatenated. Therefore, as one of the options to circumvent the over-smoothing problem, we recommend empirically checking whether the iteration affects over-smoothing in SVR with node features learned from GCNs.

3.4 Conclusion

In this chapter, we have investigated the effect of iterations of GCNs layers. In the WL kernel (which is the state-of-the-art graph kernel and shares the iteration process with GNNs to incorporate graph topological information), it usually maintains the good performance for large number of iterations in both classification and regression tasks. This may be because the WL kernel implicitly constructs a feature vector and addition of iteration means the concatenation, not the summation, of features. Since the large number of iterations means that the corresponding method tries to capture larger subgraphs, it can become insignificant if only local graph information is relevant. In GNN models such GCNs and GINs, by contrast, the number of iterations is largely affected because there are more and more trainable parameters and it is often difficult to learn them properly. Another reason could be that since node features can include noise, message passing iteration contributes to accumulate noise, resulting in worsening the performance of even the WL kernel. Since the predictive power of GCNs and GINs is comparable to that of the WL graph kernel, GNNs with mini-batch training can be an effective choice on larger dataset, as it is much more efficient than the WL kernel that requires the computation of the full kernel matrix. In our future work, we will investigate other methods such as attention-based graph neural networks.

Table 3.1: Statistics of benchmark datasets.

Name	**Statistics**			**Label/Attributes**	
	Num. of Graphs	Avg. Nodes	Avg. Edges	Node Labels	Node Attr. Dim.
MUTAG	188	17.93	19.79	+	7
PTC-MR	344	4.29	14.69	+	19
NCI1	4110	29.87	32.30	+	37
PROTEINS	1113	39.06	72.82	+	3
DD	1178	284.32	715.66	+	88
REDDIT-BINARY	2000	429.63	497.75	-	-
IMDB-BINARRY	1000	19.77	96.53	-	-
ESOL	1128	13.29	13.68	+	74
FreeSolv	642	8.72	8.39	+	74
Lipophilicity	4200	27.04	29.5	+	74

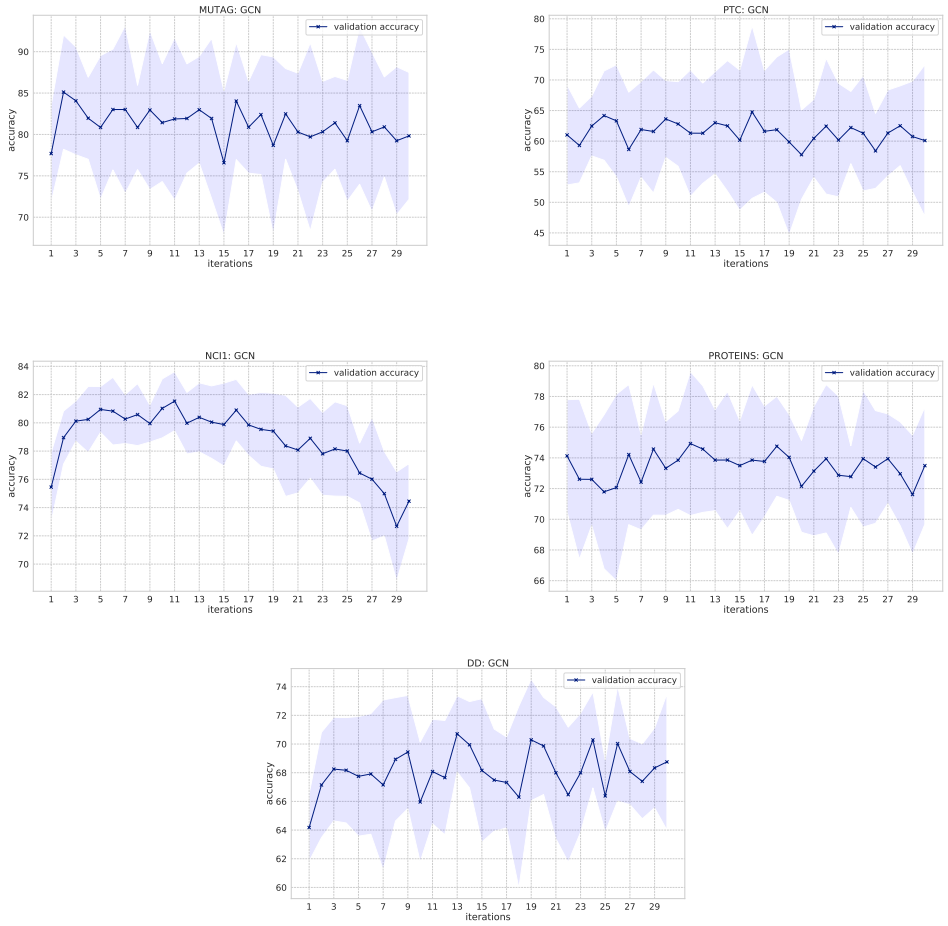


Figure 3.1: Classification results of GCNs on bio-informatics datasets. X-axis shows the number of convolutional layers (the number of message passing). Blue bold line shows the average of accuracy evaluated in 10-fold cross validation and the filled area shows the standard deviation.

Table 3.2: Classification accuracy of 10-folds cross-validation on Bio-informatics datasets.

Method	Dataset				
	MUTAG	PTC-MR	NCI1	PROTEINS	DD
V	81.96 +/- 7.06	55.82 +/- 0.91	63.24 +/- 1.83	72.42 +/- 2.53	77.85 +/- 3.11
WL	88.30 +/- 5.13	**65.68 +/- 7.44**	**86.11 +/- 1.09**	**75.29 +/- 2.52**	78.78 +/- 2.41
GRAPHLET	**89.36 +/- 4.78**	64.55 +/- 8.13	83.19 +/- 1.79	73.95 +/- 1.79	**79.03 +/- 2.91**
pre-fixed GCN SVM	87.16 +/- 6.09	59.89 +/- 3.27	64.28 +/- 2.19	71.53 +/- 4.63	75.12 +/- 2.77
GCN	85.12 +/- 6.91	64.76 +/-15.18	81.53 +/- 4.38	74.93 +/- 6.07	73.43+/- 5.56
GIN	88.89 +/- 9.80	62.20 +/-12.67	80.85 +/- 14.55	75.03+/- 8.45	77.42 +/- 9.23

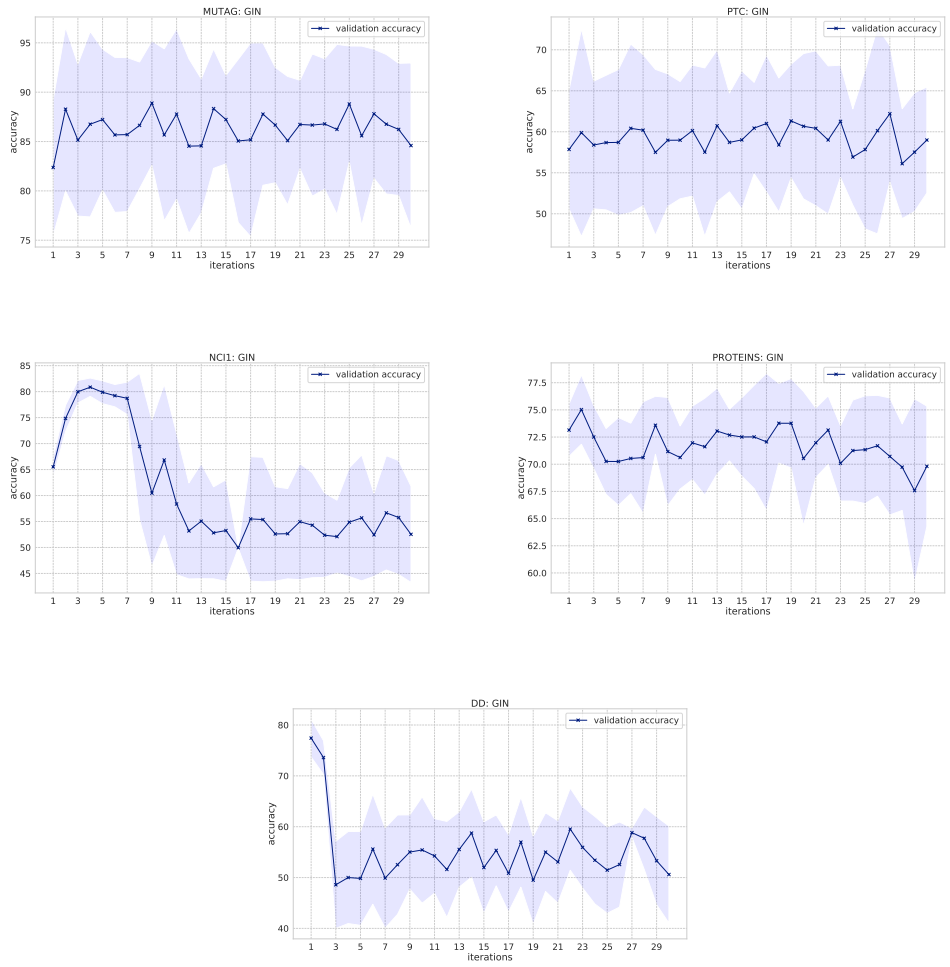


Figure 3.2: Classification results of GINs on the bio-informatics dataset.

Table 3.3: Classification accuracy of 10-folds cross-validation on social network datasets.

Method	**Dataset**	
	REDDIT-BINARY	IMDB-BINARY
V	76.90 +/- 2.10	72.6 +/- 4.27
WL	**85.61 +/- 4.87**	**73.6 +/- 4.10**
GRAPHLET	84.60 +/- 1.70	70.90 +/- 5.26
pre-fixed GCN SVM	82.55 +/- 2.60	55.50 +/- 2.25
GCN	73.43 +/- 4.63	67.00 +/- 4.42
GIN	64.90 +/- 13.35	71.00 +/- 5.29

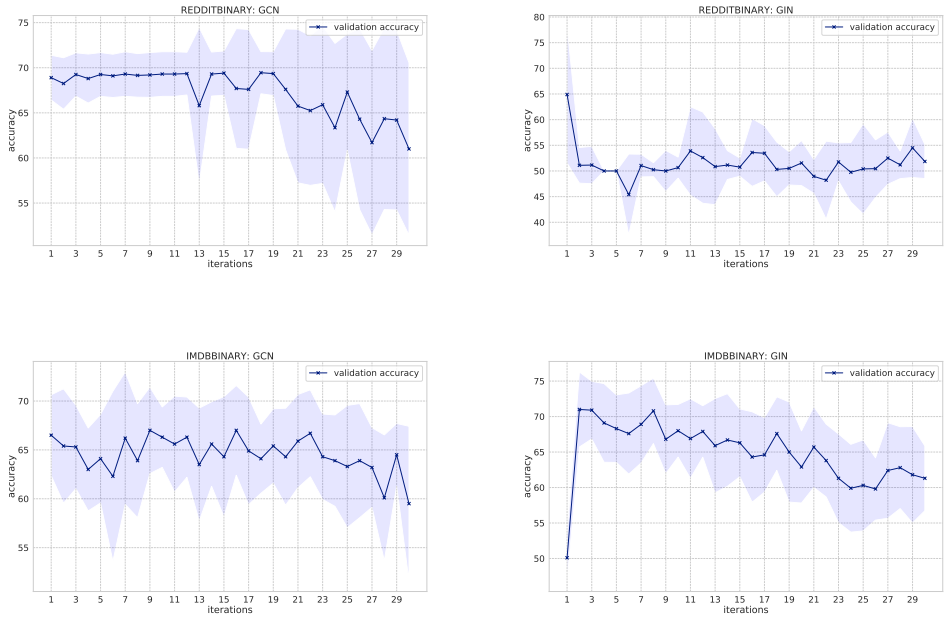


Figure 3.3: Classification results on social network datasets.

Table 3.4: RMSE of holdout validation (training 70% test 30%). "-" denotes that a model can not be trained properly under this hyperparameter.

Method	**Dataset**		
	ESOL	Free Solv	Lipophilicity
0-SVR	1.127	1.951	1.019
WL	**0.768**	1.112	**0.6074**
pre-fixed GCN SVR	1.534	3.756	1.169
1-2 layers GCN + SVR	0.8675	**0.9413**	0.6881
2 layers GCN	1.270	1.303	0.8087
2 layers GIN	0.8165	1.438	0.8118
30 layers GCN	1.016	1.407	1.219
30 layers GIN	-	14.45	-

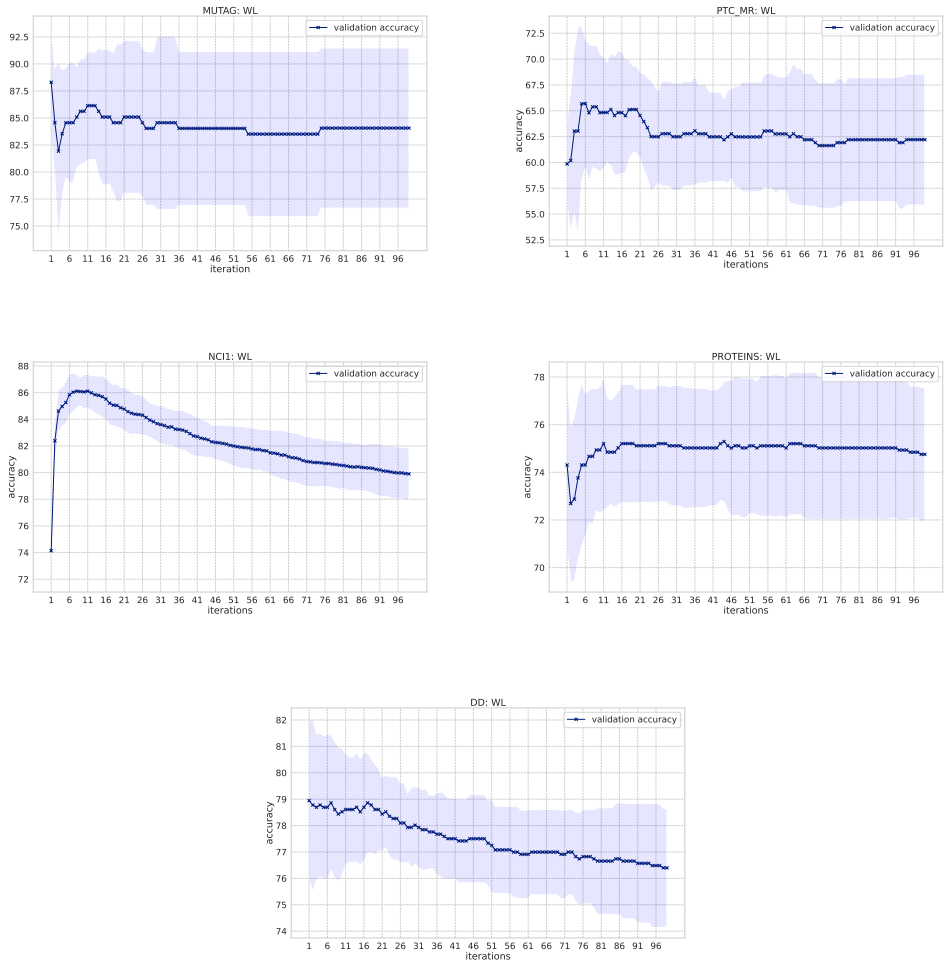


Figure 3.4: Classification results of the WL kernel on Bio-informatics datasets.

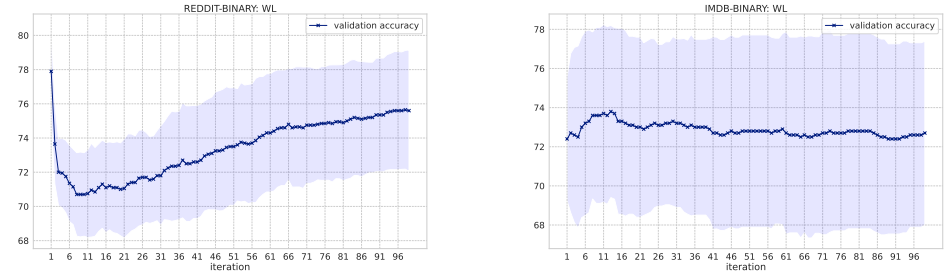
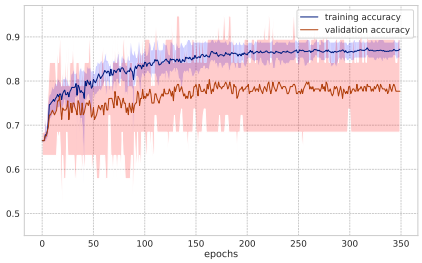
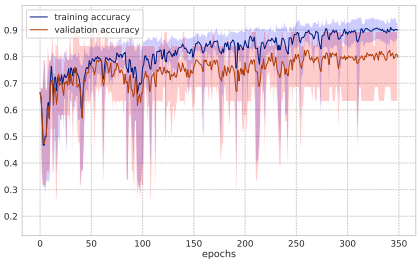


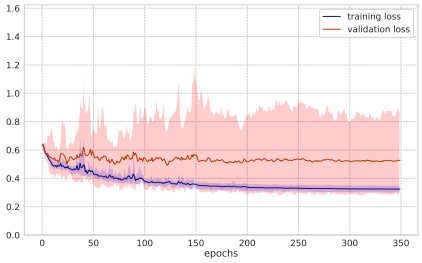
Figure 3.5: Classification results on social network datasets.



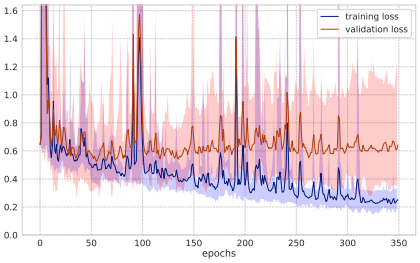
(a) Accuracy of 1-layer GCN



(b) Accuracy of 30-layers GCN

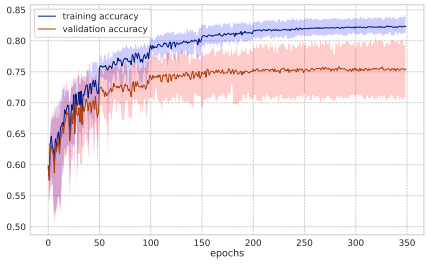


(c) Loss of 1-layers GCN

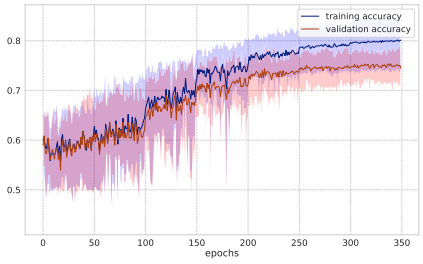


(d) Loss of 30-layers GCN

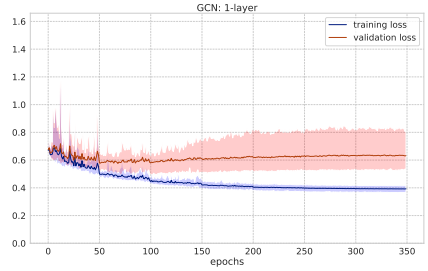
Figure 3.6: Accuracy and loss on MUTAG dataset.



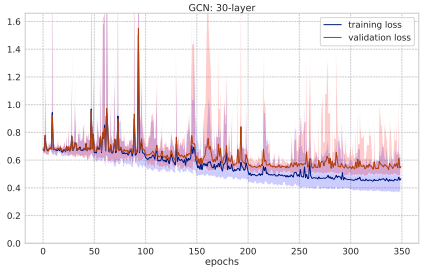
(a) Accuracy of 1-layer GCN



(b) Accuracy of 30-layers GCN

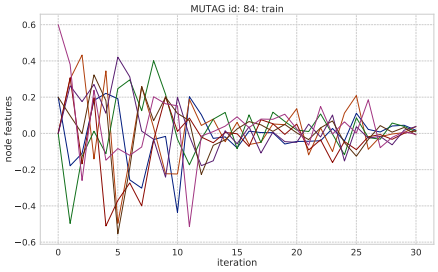


(c) Loss of 1-layers GCN

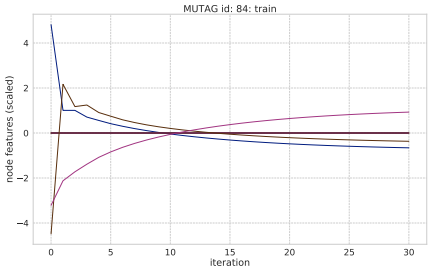


(d) Loss of 30-layers GCN

Figure 3.7: Accuracy and loss on NCI1 dataset.

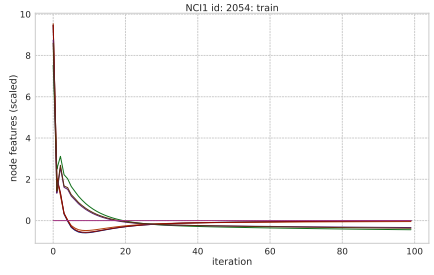


(a) GCN trained

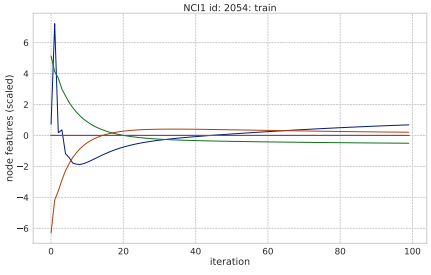


(b) Pre-fixed weights GCN

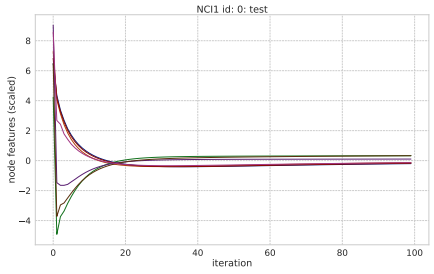
Figure 3.8: An example of transition of node features on MUTAG dataset. Each curve represents the averaged node feature value in a graph at each iteration step.



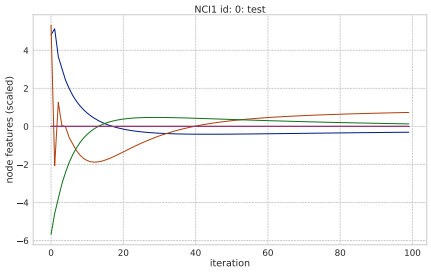
(a) Trained GCNs



(b) Pre-fixed weights GCNs



(c) Trained GCNs



(d) Pre-fixed weights GCNs

Figure 3.9: An example of transition of node features on NCI1 dataset. Each curve represents the averaged node feature value in a graph at each iteration step.

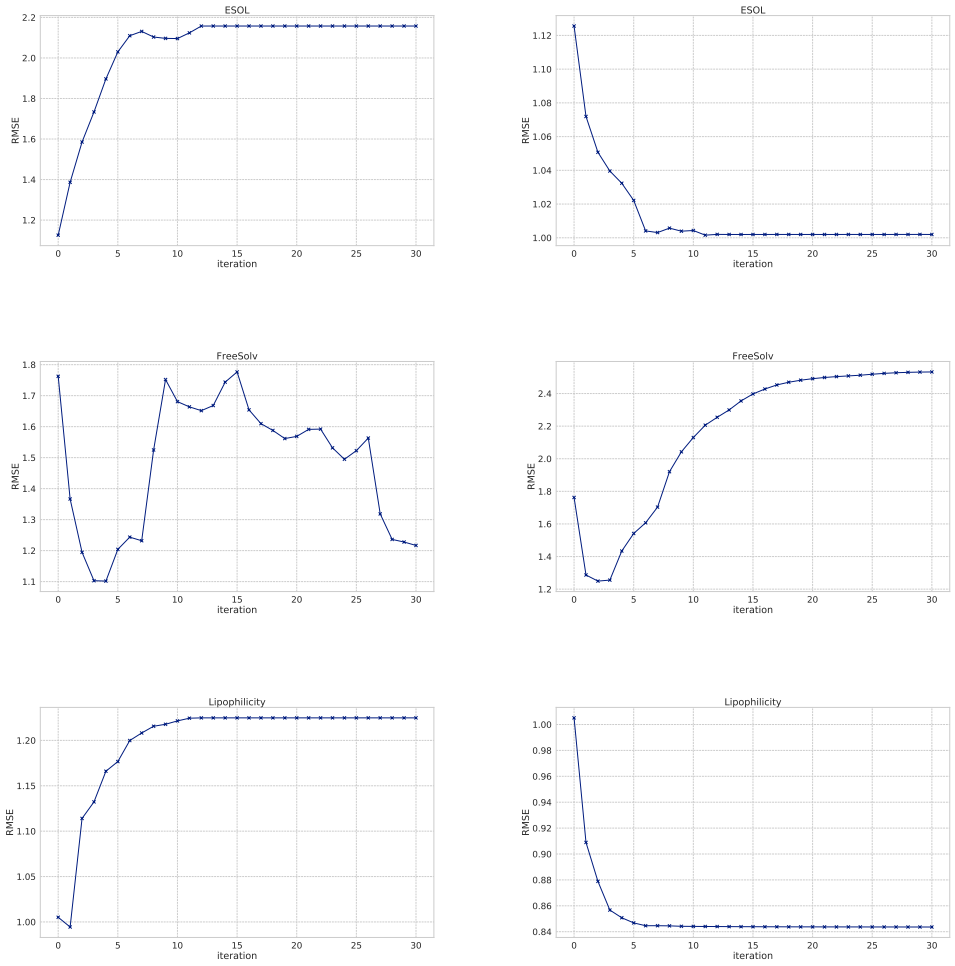


Figure 3.10: Regression results of SVR using node features learned from GCNs. Left plots show SVR results using features produced from each GCNs layer and right plots show SVR results using concatenating features.

Chapter 4

Summary and Future works

In this thesis, we have proposed a novel molecular graph generation algorithm called MOLDR (Molecular graph Decomposition and Reassembling) and investigated the effect of the message passing algorithm, the fundamental procedure of incorporating graph topological structure, as the number of iteration increase. In Chapter 1, we have described the fundamental theories in multi-disciplinary areas in order to understand graph generation and graph machine learning, especially focusing on subgraphs structures. In Chapter 2, we proposed MOLDR and showed the effectiveness of MOLDR in the task of molecular graph generation using real-world datasets. MOLDR does not need to use generative models; hence, it is not need to consider the representation of graphs when generating molecules. Due to reassembling molecules from building blocks obtained from subgraph mining, it enables chemical space to be restricted in order to prevent the combinatrial problem. In Chapter 3, we have investigated how graph features from message passing affect graph classification and regression. Although Chapter 3 is a general topic of message passing algorithms, it is related to molecular graph generation (Chapter 2) when estimating the properties of molecular graphs using graph neural networks. In order to design molecules with desired properties, an appropriate reward function is also needed to maximize the cumulative rewards. Finally, in this chapter, we summarise this thesis, current limitations, and future works.

4.1 Summary

4.1.1 Molecular Graph Generation

We have proposed a new graph generation algorithm, called *Molecular graph Decomposition and Reassembling* (MOLDR). The procedures are as follows:

1. Apply subgraph mining algorithm gSpan to a given graph dataset,

2. Select subgraph structures as building blocks,
3. Compute the target property of reassembled graphs by some target function e.g. graph regression model,
4. Apply MCTS (or reinforcement learning),
5. Compute the PUCT score against the generated molecules,
6. Expand the path for the next generated graph,
7. Update PUCT score,
8. Run until the preset condition is satisfied and back to 2.

The properties of MOLDR are as follows:

- MOLDR explicitly constructs new molecules by combining substructures, hence its generation process is interpretable.
- MOLDR can easily generate larger size of molecules out of distribution in a dataset by combining subgraph structures.
- Molecules generated by MOLDR are superior to those by the current state-of-the-art methods in terms of $\log P$ and QED (drug-likeness).

4.1.2 How Graph Features from Message Passing Affect Graph Classification and Regression

We have investigated the prediction performance of graph regression and classification regarding the WL kernel and graph neural networks with respect to the number of message passing iterations.

- We show that the prediction performance of the WL kernel outperforms the most fundamental GNNs, GCNs, and GINs, if the number of message passing iterations increases.
- The WL kernel does not significantly deteriorate for many message passing iterations in most datasets. At the same time, GCNs and GINs do due to their large number of parameters and ill-trained previous node features.
- We also show that the transition of node features tends to be small, which leads to the difficulty of determining which feature is informative for prediction. The features become certain values when the training fails.

4.2 Future Works

The graph generation task is still challenging due to the graph combinatorial problem when the number of nodes increases. Generating graphs with desired properties need an appropriate balance between exploration and exploitation. In MOLDR, generated graphs depend on the previous nodes; thus, if the previous path is not accurate, a massive rollout with higher exploration is needed to prevent this problem. MOLDR implemented with MCTS is computed with currently only a single core. In order to search efficiently, the implementation of parallel computation is also necessary. We need to investigate MOLDR further in the case of multi-objective targets and another benchmark, such as GuacaMol that is used for goal-directed molecular graph generation.

We also need to investigate the relationship between the performance and node embedding in molecular graph generation as the graph size increases. As shown in Chapter 3, as the number of message passing iterations increases, it is likely to decrease the performance of regression and classification, which means that it is difficult to take the effect of larger substructures into consideration in GCN and GIN. This would have the possibility to affect the performance of molecular graph generation since it cannot represent graph structures appropriately. We need to conduct further experiments and improve the performance of the molecular graph generation algorithm to use an appropriate embedding when applying graph neural networks and reinforcement learning.

Appendix A

Update Process of MOLDR

A.1 Merge Nodes and Edges

The procedure of merging node and edge is shown in Fig A.1 and A.2. In order to combine two graphs, two adjacency matrices should be concatenated. The process of merging nodes and edges is operated through the adjacency matrix.

In Fig. A.1, two graphs are merged with nodes by adding the bit in adjacency matrix and by removing duplicated nodes. Fig. A.2 shows that two graph G_1 and G_2 is merged into one graphs with two rings. C_6 and C_7 is overwritten into C_0 and C_1 in this case so that a bit in the adjacency matrix is add at the position that is merged. Unneeded nodes are removed in this process.

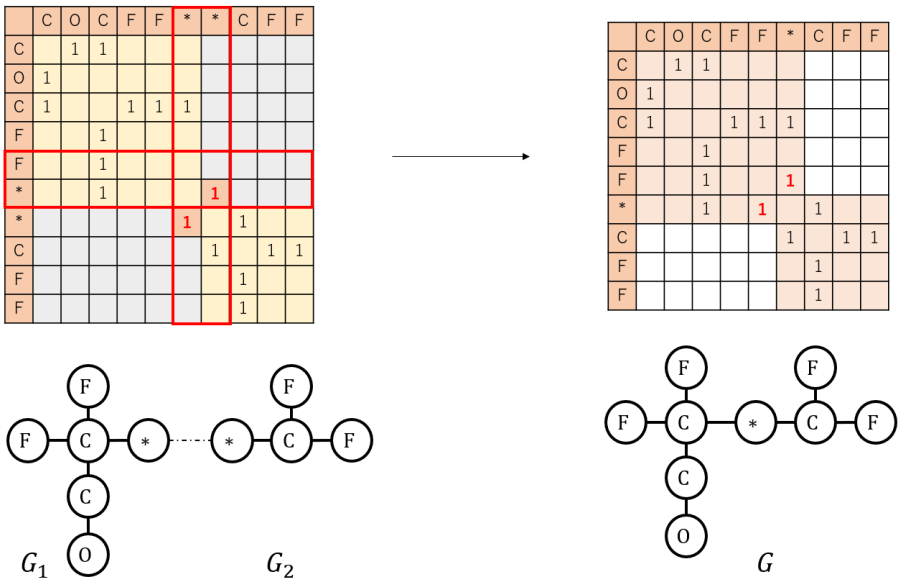


Figure A.1: Procedure of merging node between two graphs. Left figure shows the two graphs represented by one graph. Red rectangle encircling $(F, *)$ is merged from dotted reg rectangle encircling $(*, *)$. Right figure shows the result of reassembling two graphs. After merging nodes, the bits written in red number is added into adjacency matrix and the index of merged node $*$ is removed from row and column.

A.2 MOLDR with reinforcement learning

We evaluate MOLDR with reinforcement learning to compute asynchronously. To do so, we tried policy proximal optimization algorithm to train the agent of MOLDR on the GuacaMol benchmark. We evaluate the task of Troglitazone rediscovery.

A.3 How to select molecules

In reassembling process, the combination of subgraphs increases as the number of nodes increases. We check the performance how the reassembled molecules are selected with reinforcement learning. After reassembling molecules, there are multiple candidates generated through MOLDR, and there are multiple choices to select them: Random selection, Maximum score selection, and selection based on chemical reaction property. We tried random selection and maximum score selection. The reward for reinforcement learning is defined as the similarity between a generated molecule and target molecule.

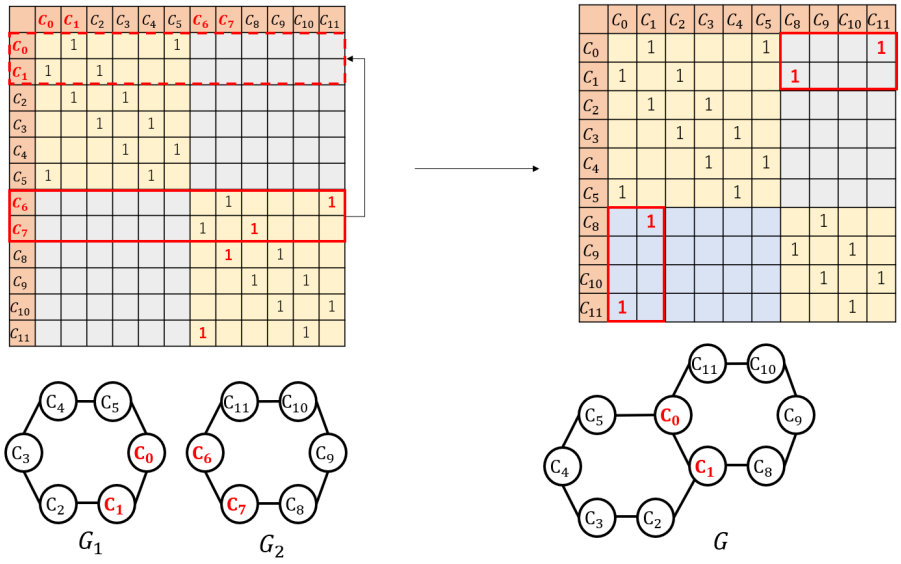


Figure A.2: Procedure of merging an edge between two graphs. Left figure shows the two graphs represented by one graph. Red rectangle encircling $C_6, C7$ is merged from dotted reg rectangle encircling C_0, C_1. Right figure shows the result of reassembling two graphs. After merging edges, the bits written in red number is added into adjacency matrix.

Random selection. Fig. A.4 shows the training results. In the case of random selection of candidates reassembled through MOLDR, the reward mean is better when the policy is MLPs. However, the max reward is almost same as the LSTMs, and MOLDR cannot reconstruct Troglitazone based on the score.

Maximum score selection. In the case of maximum score selection of candidates shown in Fig A.5, the reward mean is better when the selection is random. Generated molecules are almost same as the target molecule. However, the max reward is almost same as the LSTM, and MOLDR cannot reconstruct Troglitazone shown in Fig. A.6. Computational time is about 40 minutes with 32 CPUs and 2 GPUs.

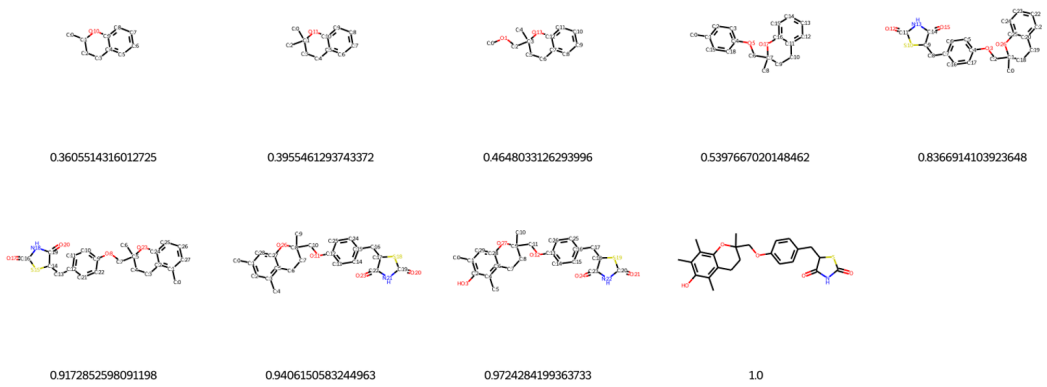


Figure A.3: Troglitazone rediscovery with MOLDR when generating successfully. The value under molecules is similarity score between a generated molecule and target molecule. Gradually molecules are reassembled and generate the final product.

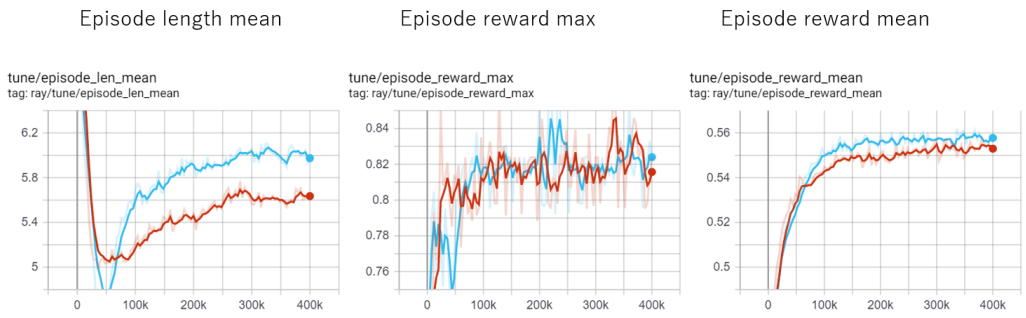


Figure A.4: Training results of MOLDR with random select when reassembling molecules. Expanding path is selected through PPO where the policy network is either LSTM (red) or MLPs (blue). In the left figure, vertical axis shows the episode length. The middle figure shows the maximum reward in the episode. Right figure shows reward mean of episode.

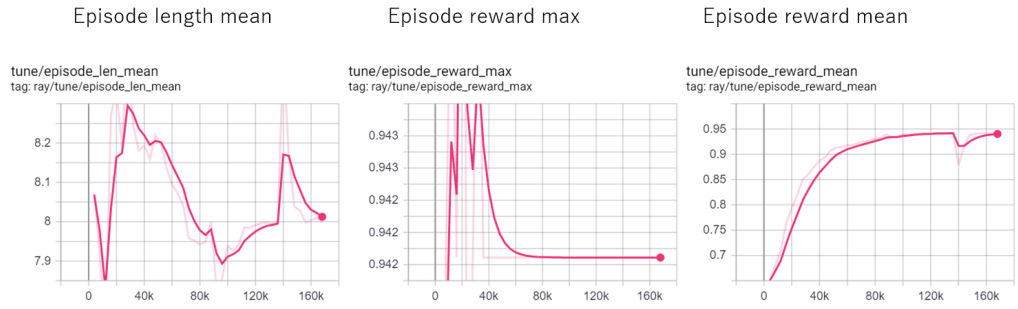


Figure A.5: Training results of MOLDR with the maximum score selection. Expanding path is selected through PPO where the policy network is MLPs.

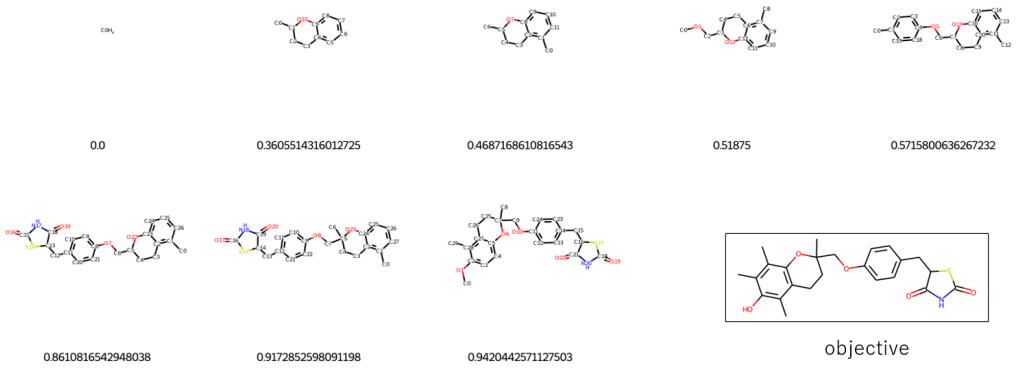
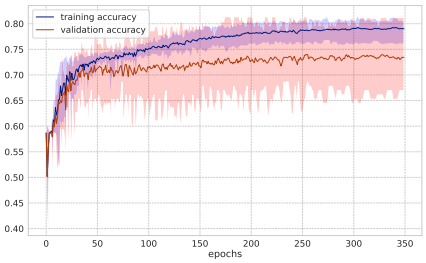


Figure A.6: Troglitazone rediscovery with MOLDR under the PPO + maximum reward selection.

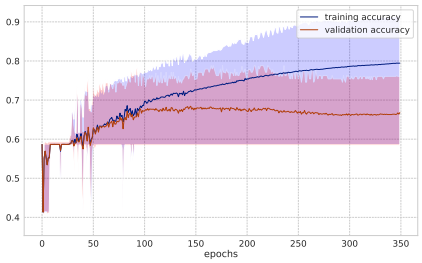
Appendix B

Experimental results for GNNs on another dataset

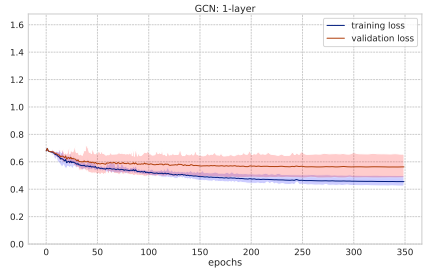
Experimental results of accuracy and loss in GCNs and GINs are shown here. The rests of dataset are DD, PROTEINS, PTC, IMDB-BINARY, and REDDIT-BINARY. As shown in here, training results of GINs is not stable when the number of message passing iteration increases, so that loss values in some figures are not shown due to the bigger values.



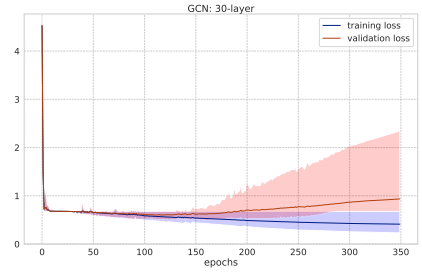
(a) Accuracy of 1-layer GCN



(b) Accuracy of 30-layers GCN

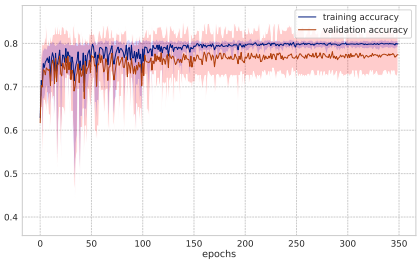


(c) Loss of 1-layers GCN

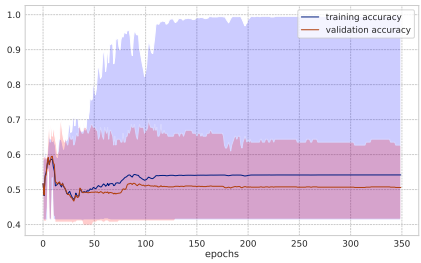


(d) Loss of 30-layers GCN

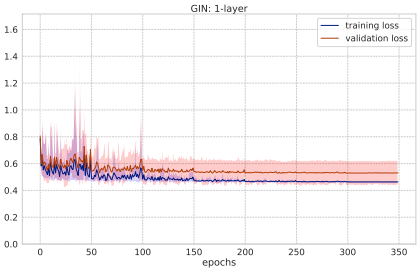
Figure B.1: GCN: Accuracy and loss on DD dataset.



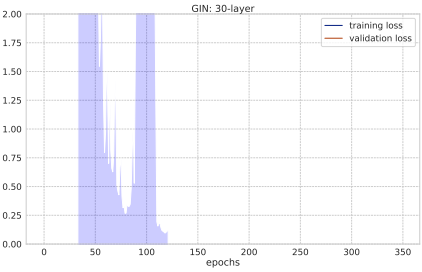
(a) Accuracy of 1-layer GIN



(b) Accuracy of 30-layers GIN

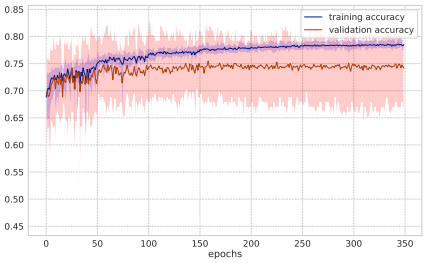


(c) Loss of 1-layers GIN

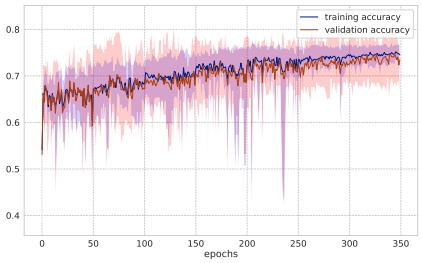


(d) Loss of 30-layers GIN

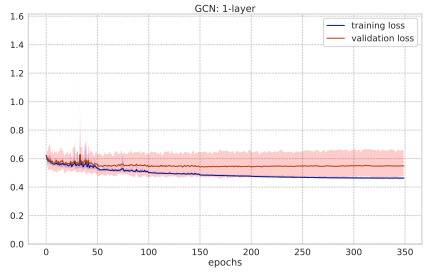
Figure B.2: GIN: Accuracy and loss on DD dataset.



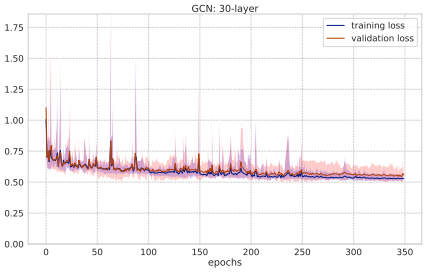
(a) Accuracy of 1-layer GCN



(b) Accuracy of 30-layers GCN

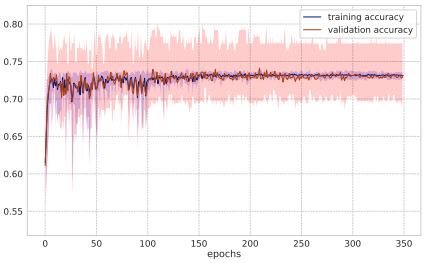


(c) Loss of 1-layers GCN

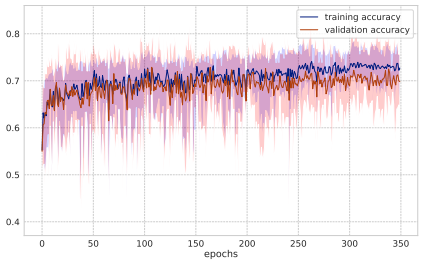


(d) Loss of 30-layers GCN

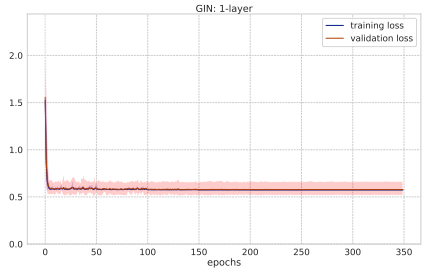
Figure B.3: GCN: Accuracy and loss on PROTEINS dataset.



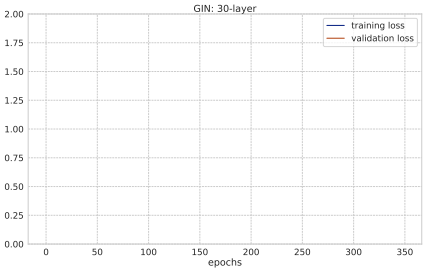
(a) Accuracy of 1-layer GIN



(b) Accuracy of 30-layers GIN

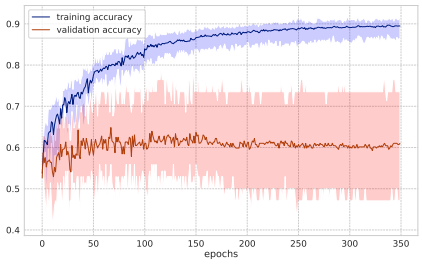


(c) Loss of 1-layers GIN

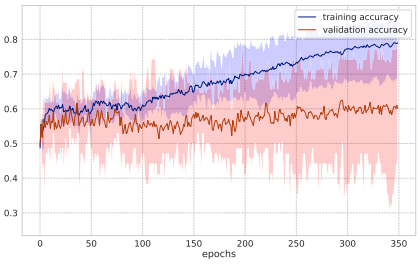


(d) Loss of 30-layers GIN

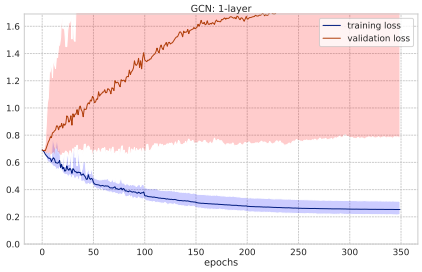
Figure B.4: GIN: Accuracy and loss on PROTEINS dataset.



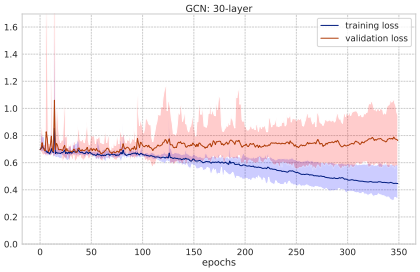
(a) Accuracy of 1-layer GCN



(b) Accuracy of 30-layers GCN

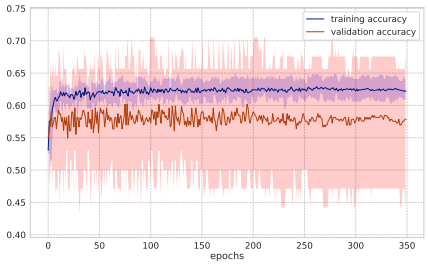


(c) Loss of 1-layers GCN

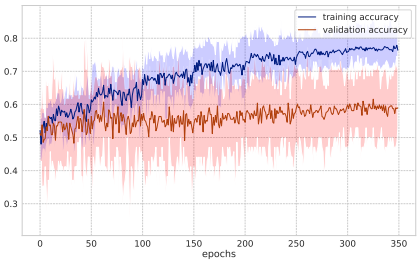


(d) Loss of 30-layers GCN

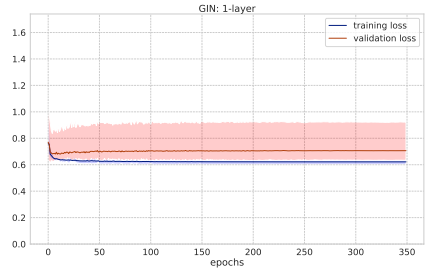
Figure B.5: GCN: Accuracy and loss on PTC dataset.



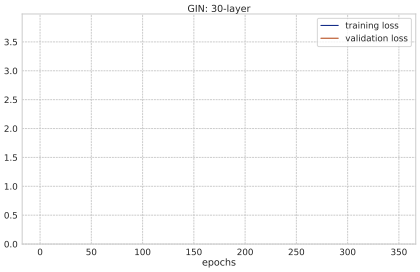
(a) Accuracy of 1-layer GIN



(b) Accuracy of 30-layers GIN

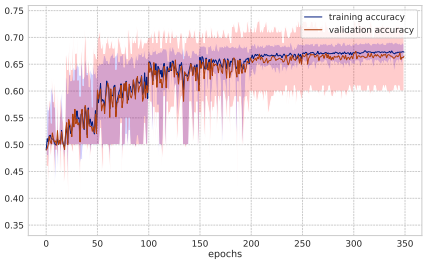


(c) Loss of 1-layers GIN

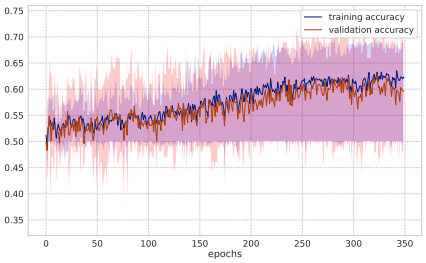


(d) Loss of 30-layers GIN

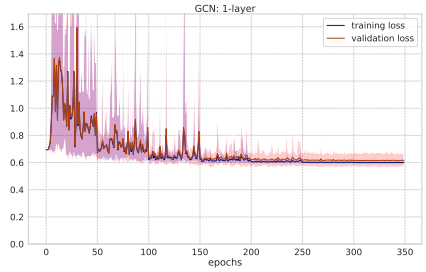
Figure B.6: GIN: Accuracy and loss on PTC dataset.



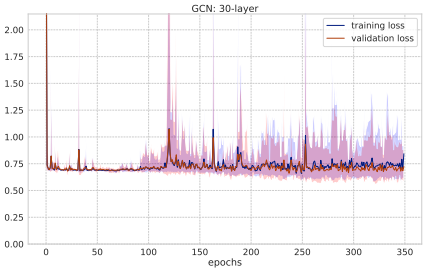
(a) Accuracy of 1-layer GIN



(b) Accuracy of 30-layers GIN

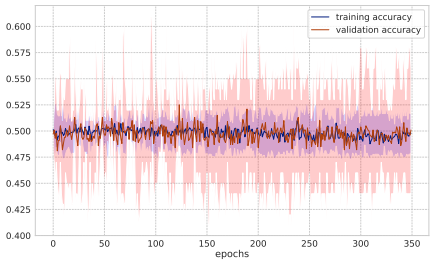


(c) Loss of 1-layer GIN

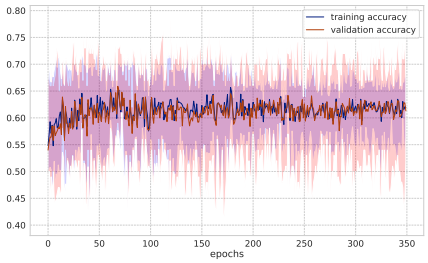


(d) Loss of 30-layers GIN

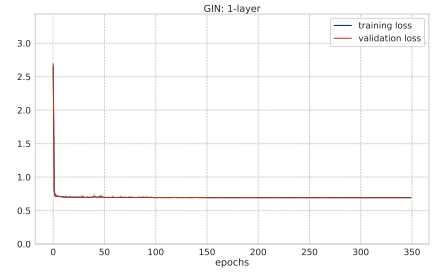
Figure B.7: GCN: Accuracy and loss on IMDBBINARY dataset.



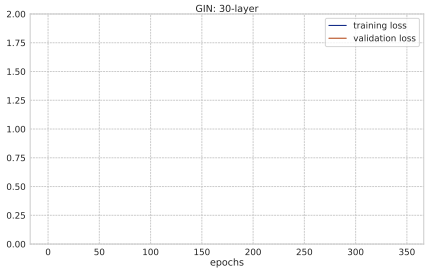
(a) Accuracy of 1-layer GIN



(b) Accuracy of 30-layers GIN

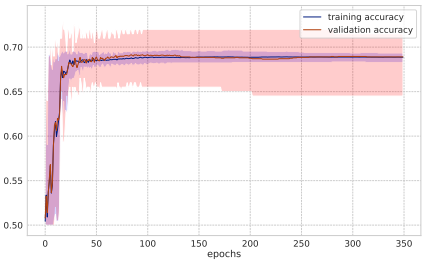


(c) Loss of 1-layers GIN

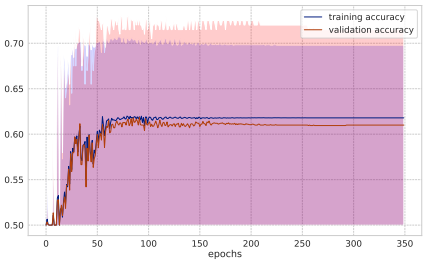


(d) Loss of 30-layers GIN

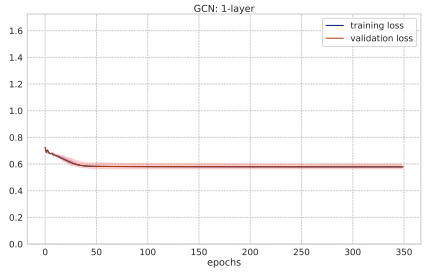
Figure B.8: GIN: Accuracy and loss on IMDBBINARY dataset.



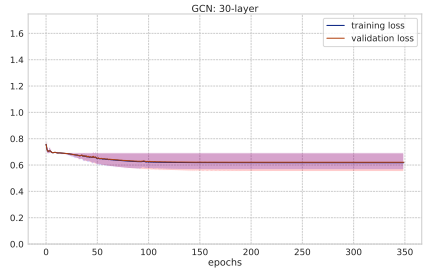
(a) Accuracy of 1-layer GCN



(b) Accuracy of 30-layers GCN

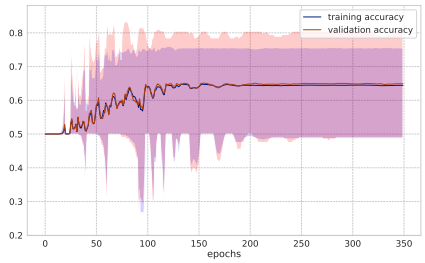


(c) Loss of 1-layer GCN

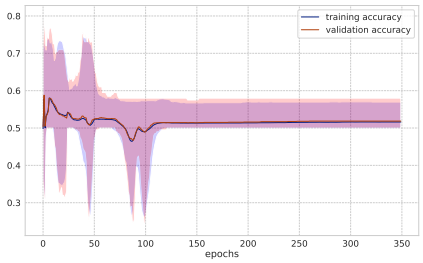


(d) Loss of 30-layers GCN

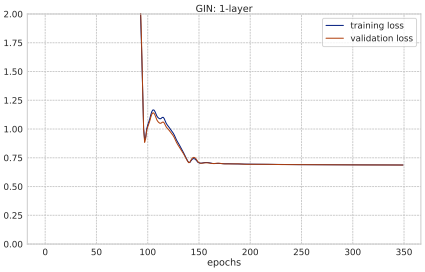
Figure B.9: GCN: Accuracy and loss on REDDITBINARY dataset.



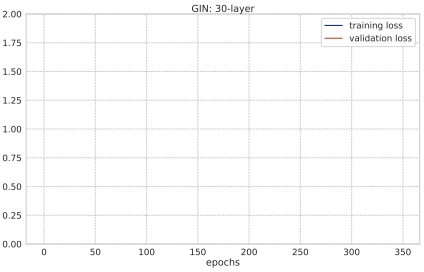
(a) Accuracy of 1-layer GIN



(b) Accuracy of 30-layers GIN



(c) Loss of 1-layers GIN



(d) Loss of 30-layers GIN

Figure B.10: GIN: Accuracy and loss on REDDITBINARY dataset.

Bibliography

Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks, 2018. URL https://arxiv.org/abs/1806.01261.

G. R. Bickerton, G. V. Paolini, J. Besnard, S. Muresan, and A. L. Hopkins. Quantifying the chemical beauty of drugs. *Nature Chemistry*, 4(2):90–98, 2012. doi: doi:10.1038/nchem.1243.

Deyu Bo, Xiao Wang, Chuan Shi, and Huawei Shen. Beyond low-frequency information in graph convolutional networks, 2021. URL https://arxiv.org/abs/2101.00797.

Karsten Borgwardt, Elisabetta Ghisu, Felipe Llinares-López, Leslie O'Bray, and Bastian Rieck. Graph kernels: State-of-the-art and future challenges. *Foundations and Trends® in Machine Learning*, 13(5-6):531–712, 2020. doi: 10.1561/2200000076. URL https://doi.org/10.1561%2F2200000076.

Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks?, 2021. URL https://arxiv.org/abs/2105.14491.

Nathan Brown, Marco Fiscato, Marwin H.S. Segler, and Alain C. Vaucher. Guacamol: Benchmarking models for de novo molecular design. *Journal of Chemical Information and Modeling*, 59(3):1096–1108, Mar 2019. ISSN 1549-9596. doi: 10.1021/acs.jcim.8b00839. URL https://doi.org/10.1021/acs.jcim.8b00839.

Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon

Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012. doi: 10.1109/TCIAIG.2012.2186810.

Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):3438–3445, Apr. 2020. doi: 10.1609/aaai.v34i04.5747. URL `https://ojs.aaai.org/index.php/AAAI/article/view/5747`.

Carla J Churchwell, Mark D Rintoul, Shawn Martin, Donald P Visco, Archana Kotu, Richard S Larson, Laurel O Sillerud, David C Brown, and Jean-Loup Faulon. The signature molecular descriptor: 3. inverse-quantitative structure–activity relationship of icam-1 inhibitory peptides. *Journal of Molecular Graphics and Modelling*, 22(4):263–273, 2004. doi: https://doi.org/10.1016/j.jmgm.2003.10.002.

Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Proceedings of the 5th International Conference on Computers and Games*, CG'06, page 72–83, Berlin, Heidelberg, 2006. Springer-Verlag.

David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL `https://proceedings.neurips.cc/paper/2015/file/f9be311e65d81a9ad8150a60844bb94c-Paper.pdf`.

Peter Ertl and Ansgar Schuffenhauer. Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions. *Journal of Cheminformatics*, 1(1):8, 2009. doi: 10.1186/1758-2946-1-8.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1263–1272. JMLR.org, 2017.

M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734 vol. 2, 2005. doi: 10.1109/IJCNN.2005.1555942.

Ralf Gugisch, Adalbert Kerber, Reinhard Laue, Axel Kohnert, Markus Meringer, Christoph Rücker, and Alfred Wassermann. *MOLGEN 5.0, A Molecular Structure Generator*, pages 113–138. BENTHAM SCIENCE, 01 2014. doi: 10.2174/978160805928711401 0010.

Gabriel Lima Guimaraes, Benjamin Sanchez-Lengeling, Pedro Luis Cunha Farias, and Alán Aspuru-Guzik. Objective-reinforced generative adversarial networks (ORGAN) for sequence generation models. *CoRR*, abs/1705.10843, 2017. URL http://arxiv.org/abs/1705.10843.

Rafael Gómez-Bombarelli, Jennifer N. Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276, Jan 2018. ISSN 2374-7951. doi: 10.1021/acscentsci.7b00572. URL http://dx.doi.org/10.1021/acscentsci.7b00572.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017.

Maarten Houbraken, Sofie Demeyer, Tom Michoel, Pieter Audenaert, Didier Colle, and Mario Pickavet. The index-based subgraph matching algorithm with general symmetries (ismags): Exploiting symmetry for faster subgraph enumeration. *Plos One*, 9(5):1–15, 05 2014. doi: 10.1371/journal.pone.0097896. URL https://doi.org/10.1371/journal.pone.0097896.

John J. Irwin, Teague Sterling, Michael M. Mysinger, Erin S. Bolstad, and Ryan G. Coleman. Zinc: A free tool to discover chemistry for biology. *Journal of Chemical Information and Modeling*, 52(7):1757–1768, 2012. doi: 10.1021/ci3001277. URL https://doi.org/10.1021/ci3001277.

Sabrina Jaeger, Simone Fulle, and Samo Turk. Mol2vec: Unsupervised machine learning approach with chemical intuition. *Journal of Chemical Information and Modeling*, 58(1):27–35, Jan 2018. ISSN 1549-9596. doi: 10.1021/acs.jcim.7b00616. URL https://doi.org/10.1021/acs.jcim.7b00616.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 2328–2337. PMLR, 2018. URL http://proceedings.mlr.press/v80/jin18a.html.

Wengong Jin, Regina Barzilay, and Tommi S. Jaakkola. Composing molecules with multiple property constraints. *CoRR*, abs/2002.03244, 2020. URL https://arxiv.org/abs/2002.03244.

Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, ICML '02, page 267–274, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc. ISBN 1558608737. URL https://people.eecs.berkeley.edu/~pabbeel/cs287-fa09/readings/KakadeLangford-icml2002.pdf.

Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. Marginalized kernels between labeled graphs. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, page 321–328. AAAI Press, 2003. ISBN 1577351894.

Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. Benchmark data sets for graph kernels, 2016. http://graphkernels.cs.tu-dortmund.de.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL https://arxiv.org/abs/1412.6980.

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

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006*, pages 282–293, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

Matt J. Kusner, Brooks Paige, and José Miguel Hernández-Lobato. Grammar variational autoencoder. *arXiv:1703.01925*, 2017.

Sunyoung Kwon, Ho Bae, Jeonghee Jo, and Sungroh Yoon. Comprehensive ensemble in qsar prediction for drug discovery. *BMC Bioinformatics*, 20(1):521, 2019. doi: 10.1186/s12859-019-3135-4.

Greg Landrum. Rdkit: Open-source cheminformatics. *Google Scholar*, 2006. URL https://www.rdkit.org/.

Mufei Li, Jinjing Zhou, Jiajing Hu, Wenxuan Fan, Yangkang Zhang, Yaxin Gu, and George Karypis. Dgl-lifesci: An open-source toolkit for deep learning on graphs in life science. *ACS Omega*, 2021.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks, 2015. URL https://arxiv.org/abs/1511.05493.

Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs, 2018. URL https://arxiv.org/abs/1803.03324.

Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. RLlib: Abstractions for distributed reinforcement learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3053–3062. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/liang18b.html.

Brendan D McKay. Isomorph-free exhaustive generation. *Journal of Algorithms*, 26(2):306–324, 1998. doi: https://doi.org/10.1006/jagm.1997.0898.

Tomoyuki Miyao, Hiromasa Kaneko, and Kimito Funatsu. Inverse qspr/qsar analysis for chemical structure generation (from y to x). *Journal of Chemical Information and Modeling*, 56(2):286–299, 2016. doi: 10.1021/acs.jcim.5b00628.

Marcus Olivecrona, Thomas Blaschke, Ola Engkvist, and Hongming Chen. Molecular de-novo design through deep reinforcement learning. *Journal of Cheminformatics*, 9(48), 2017.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

P. G. Polishchuk, T. I. Madzhidov, and A. Varnek. Estimation of the size of drug-like chemical space based on gdb-17 data. *Journal of Computer-Aided Molecular Design*, 27(8):675–679, 2013.

Kristina Preuer, Philipp Renz, Thomas Unterthiner, Sepp Hochreiter, and Günter Klambauer. Fréchet chemnet distance: A metric for generative models for molecules in drug discovery. *Journal of Chemical Information and Modeling*, 58(9):1736–1741, Sep 2018. ISSN 1549-9596. doi: 10.1021/acs.jcim.8b00234. URL https://doi.org/10.1021/acs.jcim.8b00234.

Nataša Pržulj. Biological network comparison using graphlet degree distribution. *Bioinformatics*, 23(2):e177–e183, 01 2007. ISSN 1367-4803. doi: 10.1093/bioinformatics/btl301. URL https://doi.org/10.1093/bioinformatics/btl301.

David Rogers and Mathew Hahn. Extended-connectivity fingerprints. *Journal of Chemical Information and Modeling*, 50(5):742–754, May 2010a. ISSN 1549-9596. doi: 10.1021/ci100050t. URL https://doi.org/10.1021/ci100050t.

David Rogers and Mathew Hahn. Extended-connectivity fingerprints. *Journal of Chemical Information and Modeling*, 50(5):742–754, May 2010b. ISSN 1549-9596. doi: 10.1021/ci100050t. URL https://doi.org/10.1021/ci100050t.

Christopher D. Rosin. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230, 2011. doi: 10.1007/s10472-011-9258-6.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2015a. URL https://arxiv.org/abs/1502.05477.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2015b. URL https://arxiv.org/abs/1506.02438.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Marwin H. S. Segler, Thierry Kogej, Christian Tyrchan, and Mark P. Waller. Generating focused molecule libraries for drug discovery with recurrent neural

networks. *ACS central science*, 4(1):120–131, 2018a. doi: 10.1021/acscentsci.7b00512.

Marwin H. S. Segler, Mike Preuss, and Mark P. Waller. Planning chemical syntheses with deep neural networks and symbolic ai. *Nature*, 555(7698):604–610, 2018b. doi: 10.1038/nature25978.

Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In David van Dyk and Max Welling, editors, *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pages 488–495, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 16–18 Apr 2009. PMLR. URL `https://proceedings.mlr.press/v5/shervashidze09a.html`.

Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(77):2539–2561, 2011a. URL `http://jmlr.org/papers/v12/shervashidze11a.html`.

Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(77):2539–2561, 2011b. URL `http://jmlr.org/papers/v12/shervashidze11a.html`.

Giannis Siglidis, Giannis Nikolentzos, Stratis Limnios, Christos Giatsidis, Konstantinos Skianis, and Michalis Vazirgiannis. Grakel: A graph kernel library in python. *Journal of Machine Learning Research*, 21(54):1–5, 2020.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. doi: 10.1038/nature16961.

Hartke G. Stephen and Radcliffe J. Andrew. Mckay's canonical graph labeling algorithm. *In Communicating Mathematics*, 479:99–111, 2009.

Mahito Sugiyama and Karsten Borgwardt. Halting in random walk kernels. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates,

Inc., 2015. URL https://proceedings.neurips.cc/paper/2015/file/31b3b31a1c2f8a370206f111127c0dbd-Paper.pdf.

Seiji Takeda, Toshiyuki Hama, Hsiang-Han Hsu, Victoria A. Piunova, Dmitry Zubarev, Daniel P. Sanders, Jed W. Pitera, Makoto Kogoh, Takumi Hongo, Yenwei Cheng, Wolf Bocanett, Hideaki Nakashika, Akihiro Fujita, Yuta Tsuchiya, Katsuhiko Hino, Kentaro Yano, Shuichi Hirose, Hiroki Toda, Yasumitsu Orii, and Daiju Nakano. Molecular inverse-design platform for material industries. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery; Data Mining*, page 2961–2969, 2020. doi: 10.1145/3394486.3403346.

Youhai Tan, Lingxue Dai, Weifeng Huang, Yinfeng Guo, Shuangjia Zheng, Jinping Lei, Hongming Chen, and Yuedong Yang. Drlinker: Deep reinforcement learning for optimization in fragment linking design. *Journal of Chemical Information and Modeling*, Nov 2022. ISSN 1549-9596. doi: 10.1021/acs.jcim.2c00982. URL https://doi.org/10.1021/acs.jcim.2c00982.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

S.V.N. Vishwanathan, Nicol N. Schraudolph, Risi Kondor, and Karsten M. Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11(40):1201–1242, 2010. URL http://jmlr.org/papers/v11/vishwanathan10a.html.

Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

David Weininger. SMILES, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences*, 28(1):31–36, 1988.

Boris Weisfeiler and AA Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16, 1968.

William WL Wong and Forbes J. Burkowski. A constructive approach for discovering new drug leads: Using a kernel methodology for the inverse-qsar problem. *Journal of Cheminformatics*, 1(4), 2009.

Zhenqin "Wu, Bharath Ramsundar, Evan N. Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S. Pappu, Karl Leswing, and Vijay" Pande. "moleculenet: a benchmark for molecular machine learning". *"Chem. Sci."*, "9":"513–530", "2018". doi: "10.1039/C7SC02664A". URL "http://dx.doi.org/10.1039/C7SC02664A".

Jiawei Han Xifeng Yan. gspan: Graph-based substructure pattern mining. *International Conference on Data Mining*, pages 721–724, 2002. doi: 10.1109/ICDM.2002.1184038.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km.

Xiufeng Yang, Jinzhe Zhang, Kazuki Yoshizoe, Kei Terayama, and Koji Tsuda. ChemTS: an efficient python library for de novo molecular generation. *Science and Technology of Advanced Materials*, 18(1):972–976, 2017. doi: 10.1080/14686996.2017.1401424.

Xiufeng Yang, Tanuj Aasawat, and Kazuki Yoshizoe. Practical massively parallel monte-carlo tree search applied to molecular design. In *International Conference on Learning Representations*, 2021.

Jiaxuan You, Bowen Liu, Zhitao Ying, Vijay Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, page 6412–6422, 2018.

M. J. Zaki and W. Meira, Jr. *Data Mining and Analysis: Fundamental Concepts and Algorithms*. Cambridge University Press, 2014.

Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey, 2018. URL https://arxiv.org/abs/1812.04202.