DEEP GENERATIVE MODELS FOR DRUG DESIGN

A Dissertation
Presented to
The Academic Faculty

By

Tianfan Fu

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
Computer Science
School of Computational Science and Engineering

Georgia Institute of Technology

May 2023

DEEP GENERATIVE MODELS FOR DRUG DESIGN

Thesis committee:

Dr. Jimeng Sun
School of Computational Science & Engineering
Georgia Institute of Technology

Dr. Xiuwei Zhang
School of Computational Science & Engineering
Georgia Institute of Technology

Dr. Connor Coley
School of Chemical Engineering
Massachusetts Institute of Technology

Dr. Marinka Zitnik
Biomedical Informatics
Harvard Medical School

Dr. Yunan Luo
School of Computational Science & Engineering
Georgia Institute of Technology

Date approved: April 17, 2023

Logic will take you from A to B. Imagination will take you everywhere.

To my parents, for their everlasting support.

ACKNOWLEDGMENTS

I would like to sincerely thank my advisor, Dr. Jimeng Sun, for giving me an opportunity to learn how to identify impactful topics, conduct research, and present research results. I want to express great gratitude to the committee members, Prof. Connor W. Coley, Prof. Yunan Luo, Prof. Xiuwei Zhang, and Prof. Marinka Zitnik, for participating in my thesis and providing insightful comments and helpful feedback. I am also very lucky to work with many excellent researchers during my Ph.D. studies, including senior researchers (Dr. Cao (Danica) Xiao, Dr. Connor W. Coley, Dr. Marinka Zitnik, Lucas M. Glass, Dr. Tian Gao and Dr. Tengfei Ma), and peer researchers (Kexin Huang, Wenhao Gao, Yuanqi Du, Yue Zhao, Xinhao Li, Hanchen Wang, Shengchao Liu, Ziming Liu, Chufan (Andy) Gao, Siddhartha Laghuvarapu). I would also like to thank all the help I received from labmates over these years: Dr. Edward Choi, Dr. Ioakeim (Kimis) Perros, Yu Jing, Dr. Shenda Hong, Dr. Kejing Yin, Dr. Siddarth Biswal, Zhenbang Wu, Chaoqi Yang, Zhen Lin, Junyi Gao, Zifeng Wang, Brandon Theodorou, Chufan (Andy) Gao, Siddhartha Laghuvarapu, Pengcheng (Patrick) Jiang, Trisha Das, Kowshika Sarker, Siddarth Madala. I thank my schoolmates and friends who have provided support and friendship: Tong Zhou, Zihao Hu, Wenqi Wei, Xiaoxiao Wu, Hu Hu, Chenyi An, Wenbo Chen, Rahul Duggal, David Kartchner, Huaxiu Yao, Xiao Wang, Terry and Joan Hensel. I also thank my advisors during my undergraduate and master studies, Prof. Kai Yu, Prof. Yanmin Qian, and Prof. Zhihua Zhang, and my labmates at that time, including Learning and Optimization group (Shenjian Zhao, Shenjian Zhao, Yujun Li, Luo Luo, Haishan Ye, Cheng Chen, Cong Xie, Kai Li, etc) and SpeechLab (Wei Deng, Yuan Liu, Tianxing He, Suliang Bu, Su Zhu, Lu Chen, Sibo Tong, Yongbin You, Bo Chen, Tian Tan, etc). To my mother and father, I could never be where I am without all your sacrifices, love, and support. This Ph.D. is dedicated to you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Machine learning in drug discovery has drawn significant attention and attracted explosive growth in drug discovery and development research. This dissertation studies the deep generative methods in drug design. Despite the rapid progress of machine learning, especially deep learning in drug discovery, the existing drug design methods remain challenging for real-world applications in both categories of the methods from different aspects, including sample efficiency and data requirement, which are summarized as follows.

- **Sample efficiency**. Existing drug optimization methods rely heavily on brute-force trial-and-error strategy and suffer from poor sample efficiency. A sample-efficient drug design method would save much time and computational resources.

- **Data efficiency**. Acquiring data labels (e.g., drugs' property) is typically laborious and time-consuming in drug discovery because it involves bioassay-based wet-lab experiments or animal models.

This dissertation focuses on addressing these challenges by enhancing/designing the following categories of deep generative models:

- **Enhancing graph-to-graph neural architecture**. Graph-to-graph neural architecture is used in drug design to translate a molecule to another similar molecule with property improvement. We design copy & refine (CORE) strategy [1] and Molecule Reward in deep generative models (MOLER) [2] that leverages policy gradient of reinforcement learning. Graph-to-graph methods are easy to train in an end-to-end manner and do not require an online oracle query. However, it suffers from data- and sample-inefficiency.

- **Self-supervised learning (SSL) for generation**. SSL can be pretrained in large unlabeled data, alleviating the high demand for labeled data. During the generation

process, SSL masks a subset of the whole drug molecule and samples the masked part based on deep neural network prediction. It can be applied to both small-molecule drugs (Multi-objective molecule sampling (MIMOSA) [3]) and biologics design (sampling method for inverse protein folding (SIPF) [4]). The pros are that self-supervised learning-based generation can quantify the uncertainty and be data-efficient. However, it suffers from sample inefficiency.

- **Differentiable programming for generation**. The discrete drug molecules are relaxed to differentiable ones in continuous space, so the gradient of the neural network can be back-propagated to update the differentiable drug molecules directly. The strategy can also be applied to both small-molecule drugs (differentiable scaffolding tree (DST) [5]) and biologics (constrained energy model (CEM) [6]). Differentiable programming is data- and sample-efficient via suppressing brute-force trial-and-error strategy. However, it still heavily relies on online oracle queries.

- **Intelligent combinatorial optimization**. Traditional combinatorial optimization methods such as genetic algorithms (GA) rely heavily on a random-walk-like exploration, which leads to unstable performance. To address this challenge, we propose a Reinforced Genetic Algorithm (RGA) that uses neural models to prioritize the profitable design steps. Intelligent combinatorial optimization suppresses random-walk behavior and enhances efficiency [7]. However, it still requires online oracle queries.

In the last chapter, we describe future works to extend the current research. First, we will build some hybrid models to inherit the advantages of multiple categories of generative methods. Second, we will conduct comprehensive experiments to systematically compare these generative methods.

CHAPTER 1
INTRODUCTION

Drug discovery is a time- and resource-consuming process. Approving a new drug usually takes over ten years and billions of dollars. It has attracted more and more attention, especially after the COVID pandemic. Designing novel drugs with desired properties is a fundamental task in drug discovery. The drug-like molecules are estimated at around 10^{60} [8]. Traditional drug discovery methods mainly rely on exhaustive searching approaches, e.g., high throughput screening (HTS), which searches over the existing drug database and is typically time- and resource-consuming.

To address this efficiency issue, machine learning methods, especially deep generative models, were proposed to scale up the process. There are several essential categories of machine learning methods in drug discovery.

One is the graph-to-graph model. The main idea of the graph-to-graph model is to leverage a continuous latent space to represent the discrete drug structure and optimize the latent embedding vectors of molecules. Thanks to the expressive power of various neural architectures, such as graph neural network [9], recurrent neural network [10], deep learning methods can represent the structured discrete drug into a fixed-dimensional latent variable. The mainstream density estimation models on drug discovery include variational autoencoder (VAE) [10, 11, 1, 12], generative adversarial network (GAN) [13], normalizing flow models [14], energy-based model [6, 4], etc.

Besides, self-supervised learning has also drawn much attention in recent years thanks to its remarkable ability to utilize unlabeled data. The main idea behind self-supervised learning is to predict a subset of the raw data conditioned on the rest. The conditional probability can be used in generation tasks to update each drug molecule component iteratively and has been applied to both small-molecule drug (Multi-constraint Molecule Sampling, MIMOSA [15])

and biologics design (sampling method for inverse protein folding (SIPF) [4]).

Further, we propose a novel class of methods based on Differentiable programming. The discrete drug molecules are relaxed to differentiable ones in continuous space, so the neural network gradient can be back-propagated to update the differentiable drug molecules directly. The strategy can also be applied to both small-molecule drugs (differentiable scaffolding tree (DST) [5]) and biologics (constrained energy model (CEM) [6]).

Last, we enhance the conventional combinatorial optimization methods by using a neural network to prioritize the promising searching branches and suppress the brute-force trial-and-error strategy. We successfully apply this strategy to the genetic algorithm for small-molecule drug design (Reinforced Genetic Algorithm, RGA) [7].

In this dissertation, we propose methodologies to tackle the challenges in drug design. In the following sections, we introduce each of those works in a problem-driven way by first highlighting the motivation behind each one. In each of the following chapters, we have provided more details of each proposed solution.

1.1 My Completed Works

In this dissertation, we propose solutions for the above challenges in drug design using the deep learning-based generation method as the framework. In the following section, we introduce proposed solutions by highlighting the motivation behind the problem.

1. Graph-to-Graph model:

    - **CORE: Automatic Molecule Optimization Using Copy and Refine Strategy**. This work was published at AAAI in 2020 (chapter 3).
    - **MOLER: Incorporate Molecule-Level Reward to Enhance Deep Generative Model for Molecule Optimization**. This work was published at TKDE in 2021 (chapter 4).

2. Self-supervised learning methods:

- **MIMOSA: Multi-constraint Molecule Sampling for Molecule Optimization**. This work was published at AAAI in 2021 (chapter 5).
- **SIPF: Sampling Method for Inverse Protein Folding**. This work was published at KDD in 2022 (chapter 6).

3. Differentiable programming:

   - **DST: Differentiable Scaffolding Tree for Molecule Optimization**. This work was published at ICLR in 2022 (chapter 7).
   - **Antibody Complementarity Determining Regions (CDRs) design using Constrained Energy Model**. This work was published at KDD in 2022 (chapter 8).

4. Intelligent combinatorial optimization:

   - **Reinforced Genetic Algorithm for Structure-based Drug Design.** This work was published at NeurIPS 2022 (chapter 9).

CHAPTER 2
RELATED WORKS

The goal of drug design is to produce novel and diverse molecular structures with desirable pharmaceutical properties for further validation. Machine learning methods, especially deep generative models can achieve this goal by navigating molecular space intelligently. Also, there are two major categories of drugs, including small-molecule and biologics (also known as macro-molecule drug). This chapter briefly reviews the existing works on deep generative models for drug design, including both small-molecule drugs and biologics.

2.1 Deep Generative Models for Small-molecule Drug Design

Existing small-molecule drug design methods can mainly be categorized as density estimation models and combinatorial optimization methods. Density estimation models construct a continuous distribution of general molecular structure with a deep network model so one can generate molecules by sampling from the learned distribution. Typical algorithms include variational autoencoder (VAE) [10, 11, 16], generative adversarial network (GAN) [17], graph-to-graph model [18, 1, 19], normalizing flow-based model [20, 21, 14, 22]. However, its performance is unsatisfactory, primarily due to the failure to train an adequate surrogate oracle. In addition, DGMs can leverage Bayesian optimization in latent spaces to optimize latent vectors and reconstruct to obtain the optimized molecules [10, 11]. However, such approaches usually require a smooth and discriminative latent space and, thus, an elaborate network architecture design and well-distributed data set. Also, as they learn the reference data distribution, their ability to explore diverse chemical space is relatively limited, as evidenced by the recent molecular optimization benchmarks [23, 24, 25].

On the other hand, combinatorial optimization methods mainly include deep reinforcement learning (DRL) [26, 27, 28], evolutionary learning (especially genetic algorithm) [29,

30], self-supervised learning [4, 15]. They both formulate molecule optimization as a discrete optimization task. Specifically, they modify molecule substructures (or tokens in a string representation [31]) locally, with an oracle score or a policy/value network to tell if they keep it. Due to the discrete nature of the formulation, most of them conduct an undirected search (random-walk behavior), while some recent ones like reinforcement learning try to guide the searching with a deep neural network, aiming to rid the random-walk nature. However, it is challenging to incorporate the learning objective target into the guided search. Those algorithms still require massive numbers of oracle calls, which is computationally inefficient during the inference time [24]. The problem is especially severe in real-world drug design because the oracle typically requires a large number of resources and time such as wet-lab experiments.

2.2 Deep Generative Models for Biologics Design

There are two fundamental tasks in protein/antibody design. One is inverse protein folding, which aims to design an amino acid sequence that can fold into the input 3D structure. Another is *de novo* protein/antibody design, whose objective is to design protein (its amino acid sequence or/and the corresponding 3D structure) from scratch. Specifically, for inverse protein folding, conditioned on the input 3D graph structure (the backbone), the goal is to recover the amino acid sequence. Most of the existing methods are based on graph-to-graph models by learning a mapping from a 3D graph structure to an amino acid and generating a single amino acid at a time. Many kinds of neural network models were leveraged/designed to learn the mapping, e.g., structured transformer [32], three-dimensional convolutional neural network (3DCNN) [33, 34], graph convolutional network (GCN) [35], joint sequence-folding embedding model [36]. For (b) *de novo* protein/antibody design are usually cast into a sequence generation problem or 3D graph generation problem. Existing methods include variational autoencoder (VAE) based methods [37, 38] and generative adversarial network (GAN) based methods [39, 40]. For antibody design, [41] designed a neural network

ensemble to backpropagate the gradient to update the amino acid sequence in continuous space; [42] proposed an *iterative refinement graph neural network* to jointly design 3D graph structures and amino acid sequences.

Part I

Graph-to-Graph

Overview My thesis begins by tackling the fundamental challenges in the graph-to-graph molecular optimization framework. The first challenge is that in the graph-to-graph molecular optimization method, the set of available substructures S is large (substructure is the basic building unit in the graph-to-graph model, which can be either a single ring or an atom), such an iterative prediction task is often inaccurate, especially for substructures that are infrequent in the training data. Our first works deal with the issue via proposing *Copy and Refine (CORE)* strategy, where copy strategy copies some substructure from the input molecule and refine strategy searches over the entire substructure space as refinement.

chapter 3: **CORE: Automatic Molecule Optimization Using Copy and Refine Strategy**. Tianfan Fu, Cao Xiao, Jimeng Sun. Association for the Advancement of Artificial Intelligence (AAAI) 2020.

The second challenge is that the existing graph-to-graph models restrict their attention to substructure-level generation without considering the entire molecule as a whole. To address this challenge, we propose Molecule-Level Reward functions (MOLER) [2] to encourage (1) the input and the generated molecule to be similar and to ensure (2) the generated molecule has a similar size (in terms of the number of substructures) to the input. The proposed method is model-agnostic and can enhance various graph-to-graph models (JTVAE [11], VTJNN [18], CORE [1]) with consistent performance gain.

chapter 4: **MOLER: Incorporate Molecule-Level Reward to Enhance Deep Generative Model for Molecule Optimization.** Tianfan Fu, Cao Xiao, Lucas Glass, Jimeng Sun. IEEE Transactions on Knowledge and Data Engineering (TKDE) 2021.

CHAPTER 3
CORE: AUTOMATIC MOLECULE OPTIMIZATION USING COPY & REFINE STRATEGY

3.1 Research Challenge

Molecule optimization is about generating molecule Y with more desirable properties based on an input molecule X. The state-of-the-art approaches partition the molecules into a large set of substructures S and grow the new molecule structure by iteratively predicting which substructure from S to add. Table Table 3.1 shows some data statistics about the comparison between input and target molecules on 4 datasets/tasks. From real data, we observe: **Stable principle**: Row 1 shows the percentage of original substructures in the target molecule, which is about 80% or more, and indicates many original substructures are kept in the newly generated targets. **Novelty principle**: Row 2 shows the percentage of targets that have any new substructures that do not belong to the input molecule, which is also high and indicates the need to include new substructures in the targets. Row 3 lists the number of all the substructures, i.e., $|S|$, and Row 4 lists the average substructures for molecules. However, since the set of available substructures S is large, as shown in Table Table 3.1, the global set of substructures are 967, 785, 780, and 780 on four optimization datasets. Such an iterative prediction task is often inaccurate, especially for substructures that are infrequent in the training data.

3.2 Main Idea and Contributions

Based on these observations, we propose a new strategy for molecular optimization called *Copy & Refine* (CORE) to address this challenge. The key idea is at each generating step, CORE will decide whether we copy a substructure from the input molecule (*Copy*) or sample

Table 3.1: Comparison between input and target molecules on four molecule optimization datasets/tasks.

	DRD2	LogP04	LogP06	QED
% of original	80.42%	79.47%	88.90%	83.32%
% of novel	86.40%	84.06%	70.14%	80.84%
# substructures	967	785	780	780
Molecule size	13.85	14.30	14.65	14.99

a novel substructure from the entire space of substructures (*Refine*). CORE achieved up to 11% and 21% relative improvement over the baselines on success rate on the complete test set and test subset with infrequent substructures set, respectively.

3.3 CORE Framework

This section presents how the copy & refine strategy enhances the graph-to-graph model. Graph-to-graph model consists of two neural network modules:

1. an **encoder** that represents the structured data into a latent variable.
2. a **decoder** that constructs the structured data based on the latent variable.

Also, both the encoder and decoder involve two important structures:

1. **molecular graph** G is the graph structure for a molecule;
2. **scaffolding tree** $\mathcal{T}_G$ (also referred to as junction trees in [18]) is the skeleton of the molecular graph G by partitioning the original graph into substructures (subgraphs) and connecting those substructures into a tree.

Table Table 3.2 lists all the important mathematical notations and their explanations used in this section. Then we elaborate on the details.

3.3.1 Encoder

To construct cycle-free structures, scaffolding tree $\mathcal{T}_G$ is generated via contracting certain vertices of G into a single node. By viewing the scaffolding tree as a graph, both input

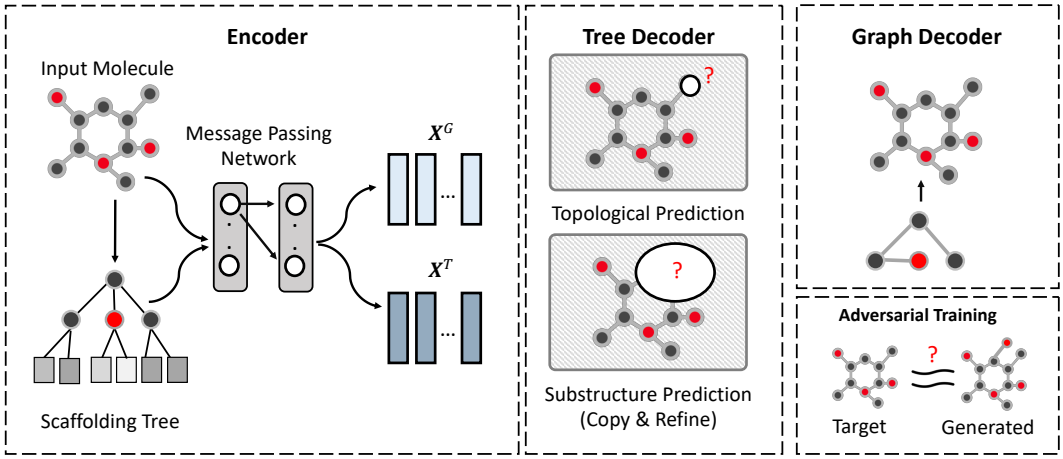


Figure 3.1: Pipeline for Graph-to-Graph Model. The encoder includes both graph and scaffolding tree levels. Decoding is mainly split into two parts scaffolding tree decoder and graph decoder. scaffolding tree decoder generates molecules in a greedy manner using Depth First Search with topological and substructure prediction on each node. To assemble the node of the scaffolding tree into the molecule, the graph decoder enumerates all possible combinations.

molecular graphs and scaffolding trees can be encoded via graph Message Passing Networks (MPN) [11, 43]. The encoder yields an embedding vector for each node in either the scaffolding tree or the input molecular graph. More formally, on the node level, $\mathbf{f}_v$ denotes the feature vector for node v. For atoms, $\mathbf{f}_v$ includes the atom type, valence, and other atomic properties. For nodes in the scaffolding tree representing substructures, $\mathbf{f}_v$ is a one-hot vector indicating its substructure index. On the other hand, on edge level, $\mathbf{f}_{uv}$ feature vector for edge $(u, v) \in E$. $N(v)$ denotes the set of neighbor nodes for node v. $\mathbf{v}_{uv}$ and $\mathbf{v}_{vu}$ are the hidden variables that represent the message from node u to v and vice versa. They are iteratively updated via a fully-connected neural network $g_1(\cdot)$:

$$\mathbf{v}_{uv}^{(t)} = g_1\big(\mathbf{f}_u, \mathbf{f}_{uv}, \sum_{w \in N(u) \setminus v} \mathbf{v}_{wu}^{(t-1)}\big), \tag{3.1}$$

Table 3.2: Important notations used in the CORE.

Notations	short explanations
(X, Y)	input-target molecule pair.
S	substructure set S (a.k.a. vocabulary).
V/E	set of vertex (atom) / edge (bond).
$G = (V, E)$	molecular graph.
$\mathcal{T}_G = (\mathcal{V}, \mathcal{E})$	scaffolding tree of graph G.
$N(v)$	set of neighbor nodes of vertex v.
$\mathbf{f}_v/\mathbf{f}_{uv}$	feature vector for node v / edge (u, v).
$\mathbf{v}_{uv}^{(t)}$	message for edge (u, v) at t-th iteration.
$\mathbf{x}_i^G/\mathbf{x}_i^{\mathcal{T}}$	embedder of node i in G / $\mathcal{T}$, $\mathbf{X}^G = \{\mathbf{x}_1^G, \cdots\}$.
$\mathbf{h}_{i_t,j_t}$	message vector for edge (i_t, j_t)
$\mathbf{z}_G$	embedding of Graph G.
p_t^{topo}	topological prediction score.
$\mathbf{q}_t^{\text{sub}}/\tilde{\mathbf{q}}_t^{\text{sub}}$	predicted distribution over all the substructures.
$g_i(\cdot),\ i = 1, \cdots, 6$	parameterized neural networks.

where $\mathbf{v}_{uv}^{(t)}$ is the message vector at the t-th iteration, whose initialization is $\mathbf{v}_{uv}^{(0)} = \mathbf{0}$. After T steps of iteration, another network $g_2(\cdot)$ is used to aggregate these messages. Each vertex has a latent vector as

$$\mathbf{x}_u = g_2\big(\mathbf{f}_u, \sum_{v \in N(v)} \mathbf{v}_{vu}^{(T)}\big), \tag{3.2}$$

where $g_2(\cdot)$ is another fully-connected neural network.

In summary, the encoder module yield embedding vectors for nodes in graph G and scaffolding tree $\mathcal{T}_G$, denoted $\mathbf{X}^G = \{\mathbf{x}_1^G, \mathbf{x}_2^G, \cdots\}$ and $\mathbf{X}^{\mathcal{T}_G} = \{\mathbf{x}_1^{\mathcal{T}_G}, \mathbf{x}_2^{\mathcal{T}_G}, \cdots\}$, respectively.

3.3.2 Decoder

Decoder has two phases in a coarse-to-fine manner: (A) tree decoder; (B) graph decoder.

A. Tree decoder. The objective of the scaffolding tree decoder is to generate a new scaffolding tree from the embeddings. The overall idea is to generate one substructure at a time from an empty tree, and at each time, we decide whether to expand the current node or backtrack to its parent (*topological prediction*) and which to add (*substructure prediction*).

The generation will terminate once the condition to backtrack from the root is reached. Tree decoder has two prediction tasks: Topological prediction and Substructure prediction.

The idea of topological prediction is to first enhance the embedding for node i_t via a tree-based RNN [11], then predict whether to expand or backtrack (binary classification).

Substructure prediction (multi-class classification) is conducted to find which substructures to add. The idea is every time expanding a new node, the model has to predict its substructure from the substructure set. First, we use the attention mechanism to compute context vector based on current message vector $\mathbf{h}_{i_t,j_t}$ and node embedding $\mathbf{X}^{\mathcal{T}}, \mathbf{X}^{G}$:

$$\mathbf{c}_t^{\text{sub}} = \text{Attention}(\mathbf{h}_{i_t,j_t}, \mathbf{X}^{\mathcal{T}}, \mathbf{X}^{G}), \tag{3.3}$$

Specifically, we first compute attention weight via

$$\begin{aligned} &\alpha_j^{\mathcal{T}} = g_4(\mathbf{h}_{k,i_t}, \mathbf{x}_j^{\mathcal{T}}), \\ &[\alpha_1^{\mathcal{T}}, \alpha_2^{\mathcal{T}}, \cdots] = \text{Softmax}([\alpha_1^{\mathcal{T}}, \alpha_2^{\mathcal{T}}, \cdots]), \end{aligned} \tag{3.4}$$

where $g_4(\cdot)$ is dot-product function [44]. $\{\alpha^G\}$ are generated in the same way. The context vector is generated via concatenating tree-level and graph-level context vector

$$\mathbf{c}_t^{\text{sub}} = \Big[\sum_i \alpha_i^{\mathcal{T}} \mathbf{x}_i^{\mathcal{T}}, \sum_j \alpha_j^{G} \mathbf{x}_j^{G}\Big]. \tag{3.5}$$

Then based on attention vector $\mathbf{c}_t^{\text{sub}}$ and message vector $\mathbf{h}_{i_t,j_t}$, $g_5(\cdot)$, a fully-connected neural network with softmax activation, is added to predict the substructure:

$$\mathbf{q}_t^{\text{sub}} = g_5(\mathbf{h}_{i_t,j_t}, \mathbf{c}_t^{\text{sub}}), \tag{3.6}$$

where $\mathbf{q}_t^{\text{sub}}$ is a distribution over all substructures. However, the number of all possible substructures is usually quite large, as shown in Table Table 3.1. The vocabulary size of the DRD2 dataset is 967, which makes prediction more challenging, especially for the rare

substructures. We design a strategy to copy some of the input sequences to the output.

Refine with novel substructures. First, we use context vector $\mathbf{c}_t^{\text{sub}}$ and embeddings of the input molecule graph and its scaffolding tree to determine the weight of generating novel substructures in the current step,

$$w_t^{\text{OOI}} = g_6(\mathbf{c}_t^{\text{sub}}, \mathbf{z}), \tag{3.7}$$

where $g_6(\cdot)$ is a fully-connected neural network with sigmoid activation. Thus, the weight ranges from 0 to 1. w_t^{OOI} represents the probability that the model generate OOI substructure at t-th step. We assume that the weight depends on the input molecule (global information) and the current position in the decoder (local information). We use a representation $\mathbf{z}$ to express the global information of the input molecule,

$$\mathbf{z} = \big[\frac{1}{|\{\mathbf{x}_i^{\mathcal{T}}\}|}\sum_i \mathbf{x}_i^{\mathcal{T}}, \frac{1}{|\{\mathbf{x}_j^{G}\}|}\sum_j \mathbf{x}_j^{G}\big], \tag{3.8}$$

where $\mathbf{z}$ is the concatenation of the average embedding of all the scaffolding tree nodes ($\mathbf{x}_j^{\mathcal{T}}$) and the average embedding of all the graph nodes ($\mathbf{x}_j^{G}$).

Copy existing substructures. After obtaining the weight of OOI substructure, we consider what substructures to copy from the input molecule. Each substructure in the input molecule has an attention weight (normalized, so the sum is 1), which measures the contribution of the substructure to the decoder. Then we use it to represent the selection probability for each substructure. Specifically, we define a sparse vector $\mathbf{a}$ as

$$\{\mathbf{a}\}_i = \begin{cases} \sum_{j\in\mathcal{C}} \alpha_j^{\mathcal{T}}, & \mathcal{C} = \{j | j\text{-th node is i-th substructure}\}, \\ 0, & \text{i-th substructure not in } \mathcal{T}, \end{cases} \tag{3.9}$$

Table 3.3: Empirical results measured by **Similarity** for various methods on different dataset.

Method	Test Set				Test subset with infrequent substructures			
	QED	DRD2	LogP04	LogP06	QED	DRD2	LogP04	LogP06
JTVAE	.2988	.2997	.2853	.4643	.2519	.2634	.2732	.4238
GCPN	.3081	.3092	.3602	.4282	.2691	.2759	.2973	.3709
Graph-to-Graph	.3145	.3164	.3579	.5256	.2723	.2760	.2901	.4744
CORE	**.3211**	**.3334**	**.3695**	**.6386**	**.2982**	**.3021**	**.3234**	**.5839**

Table 3.4: Empirical results measured by **Property Improvement**.

Method	Test Set				Test subset with infrequent substructures			
	QED	DRD2	LogP04	LogP06	QED	DRD2	LogP04	LogP06
JTVAE	.8041	.7077	2.5328	1.0323	.7292	.6292	2.0219	.7832
GCPN	.8772	.7512	3.0483	**2.148**	.7627	.6743	2.5413	1.813
Graph-to-Graph	.8803	.7641	2.9728	1.983	.7632	.6843	2.4083	1.778
CORE	**.8952**	**.7694**	**3.1053**	2.021	**.7899**	**.7193**	**2.7391**	**1.820**

where $\mathbf{a} \in \mathbb{R}^{|S|}$, $|S|$ is size of $\{\mathbf{a}\}_i$ represent i-th element of $\mathbf{a}$. The prediction is a hybrid of

$$\tilde{\mathbf{q}}_t^{\text{sub}} = w_t^{\text{OOI}} \, \mathbf{q}_t^{\text{sub}} + (1 - w_t^{\text{OOI}}) \mathbf{a}_t. \tag{3.10}$$

where w_t^{OOI} balances the contributions of two distributions at t-th step. If novel substructures are generated, we select the substructure from all substructures according to distribution $\mathbf{q}_t^{\text{sub}}$). Otherwise, we copy a certain substructure from the input molecule.

B. Graph decoder aims at assembling nodes in a scaffolding tree together into the correct molecular graph. During the learning procedure, all candidate molecular structures $\{G_i\}$ are enumerated and the learning target is to maximize the scoring function of the right structure G_o $\mathcal{L}_g = f^a(G_o) - \log \sum_{G_i} \exp(f^a(G_i))$, where $f^a(G_i) = \mathbf{h}_{G_i} \cdot \mathbf{z}_{G_o}$ is a scoring function that measures the likelihood of the current structure G_i, $\mathbf{z}_{G_o}$ is an embedding of graph G_o.

3.4 Main Results

Experimental setup. We use ZINC as the data source, which contains 250K drug molecules extracted from the ZINC database [45]. We extract data pairs from ZINC.

Table 3.5: Empirical results measured by **SR (Success Rate)** for various methods on different dataset. For QED and DRD2, when the similarity between the input and the generated molecule is greater than 0.3 and QED improvement is greater than 0.6, we regard it "success". For LogP04 and LogP06, when the similarity between input and the generated molecule is greater than 0.4 and LogP improvement is greater than 0.8, we regard it "success".

Method	Test Set				Test subset with infrequent substructures			
	QED	DRD2	LogP04	LogP06	QED	DRD2	LogP04	LogP06
JTVAE	43.32%	34.83%	38.43%	43.54%	38.91%	29.32%	35.32%	40.43%
GCPN	47.71%	44.05%	56.43%	52.82%	42.80%	37.82%	42.81%	43.29%
Graph-to-Graph	48.16%	45.73%	56.24%	55.15%	43.43%	38.39%	42.83%	47.02%
CORE	**50.26%**	**47.91%**	**56.47%**	**57.64%**	**47.82%**	**42.72%**	**45.01%**	**50.05%**

Following [18], we mainly focus on the following three properties:

- **DRD2** measures a molecule's biological activity against target dopamine type 2 (DRD2). The QED score ranges from 0 to 1. A higher value is more desirable.
- **QED** [46] is an indicator of drug-likeness. The QED value ranges from 0 to 1. A higher value is more desirable.
- **Penalized LogP** is a logP score that also accounts for ring size and synthetic accessibility [47]. The LogP values range from $-\infty$ to ∞. A higher value is more desirable.

For all these three scores, higher is better. To construct the training data set, we find the molecule pair (X, Y) following [18], where X is the input molecule and Y is the target molecule with the desired property. Both X and Y are from the whole dataset and have to satisfy two rules:

1. they are similar enough, i.e., $\text{sim}(X, Y) \geq \eta_1$;
2. Y has significant property improvement over X, i.e., $\text{property}(Y) - \text{property}(X) \geq \eta_2$, $\text{property}(\cdot)$ can be DRD2, QED, LogP score as mentioned above. $\eta_1 = 0.4$ for LogP04 while $\eta_1 = 0.6$ for LogP06.

We use the public dataset in [25] with paired data. We focus on the following evaluation metrics. Similarity: molecular similarity between the input molecule X and the generated molecule Y. Property Improvement: improvement of property scores.

Baseline Methods. We compare our method with some important baseline methods, which represent state-of-the-art methods for this task.

- **JTVAE** [11]. Junction tree variational autoencoder (JTVAE) is a deep generative model that learns latent space to generate desired molecules. It also uses encoder-decoder architecture on both scaffolding tree and graph levels.
- **Graph-to-Graph** [18]. It is the most important benchmark method as described above.
- **GCPN** [26] uses graph convolutional policy networks to generate molecular structures with specific properties. It exhibits state-of-the-art performance in RL-based methods.

Note that we also tried Sequence-to-Sequence model [48] on SMILES strings, but the resulting model generates too many invalid SMILES strings to compare with all the other graph-based methods. This further confirmed that graph generation is a more effective strategy for molecular optimization.

Evaluation. During the evaluation procedure, we mainly focus on several evaluation metrics, where X is the input molecule in the test set, Y is the generated molecule.

- **Similarity**. We evaluate the molecular similarity between the input molecule and the generated molecule, measured by Tanimoto similarity over Morgan fingerprints [49]. The similarity between molecule X and Y is denoted $\text{sim}(X, Y)$, ranging from 0 to 1.
- **Property Improvement**. The second metric is the improvement of scores on certain properties. It is defined as $\text{Property}(Y) - \text{Property}(X)$, where property could be

including QED-score, DRD2-score, and LogP-score, evaluated using Rdkit package [50].

- **Success Rate (SR)**. Success Rate is a metric that considers both similarity and property improvement. Since the task is to generate a molecule that (i) is similar to the input molecule and (ii) has property improvement at the same time. We design criteria to judge whether it satisfied these two requirements: (a) Input and generated molecules are similar enough, $\text{sim}(X, Y) \geq \lambda_1$; (b) improvement are big enough, i.e., $\text{property}(Y) - \text{property}(Y) \geq \lambda_2$. The selection of λ_1 and λ_2 depend on datasets.

Among these metrics, similarity, and property improvement are the most basic metrics. For all these metrics except run time and model size, higher values are better.

Implementation Details. We also provide the implementation details for reproducibility, especially the setting of hyperparameters. We follow most of the hyperparameter settings of [18]. For all these baseline methods and datasets, the maximal epoch number is set to 10, the batch size is set to 32. During the encoder module, the embedding size is set to 300. The depth of the message passing network is set to 6 and 3 for the tree and graph, respectively. The initial learning rate is set to $1e^{-3}$ with the Adam optimizer. Every epoch learning rate is annealed by 0.8. We save the checkpoint every epoch during the training procedure. When evaluating, from all the checkpoints, we choose the one that achieves the highest success rate (SR1) on the validation set as the best model and use it to evaluate the test set. During adversarial training, discriminator $D(\cdot)$ is a three-layer feed-forward network with hidden layer dimension 300 and LeakyReLU activation function. For all these datasets, the model size of CORE is around 4M.

Results and analysis. The results for various metrics (including similarity, property improvement, and success rate) are shown in Table Table 3.3, Table 3.4 and Table 3.5. CORE outperforms baselines in all the measures. Specifically, when measured by success

rate, CORE obtained about 2% absolute improvement over the best baseline on average and over 10% relative improvement on QED and DRD2. Test subset with infrequent substructures is more challenging because for all the methods the performance would degrade on the infrequent subset. Thus, the molecule with an infrequent substructure is worth special attention. When measured on the test subset with infrequent substructure, CORE achieves more significant improvement than the complete test set. Concretely, CORE achieves 21% and 18% relative improvement in QED and DRD2. In a word, CORE gains more improvement in rare substructures compared with the whole test set.

3.5 Conclusion and Discussion

In this work, we focus on generating molecules with desirable properties. The state-of-the-art Graph-to-Graph grows the new molecule structure by iteratively predicting substructure from a large set of substructures, which is challenging, especially for low-frequent substructures. To address this challenge, we have proposed a new generating strategy called "Copy & Refine" (COre), where at each step, the generator first decides whether to copy an existing substructure from input X or to generate a new substructure from the large set. The resulting CORE mechanism can significantly outperform several latest molecule optimization baselines in various measures, especially on rare substructures.

CHAPTER 4

MOLER: INCORPORATE MOLECULE-LEVEL REWARD TO ENHANCE DEEP GENERATIVE MODEL FOR MOLECULE OPTIMIZATION

4.1 Research Challenge

Despite the initial success of deep generative models in drug design, existing methods often rely on iterative local expansion to acquire the target molecule, potentially creating molecules of arbitrary sizes. Without a global fitness metric, the generated molecules can significantly deviate from the target molecule in molecular similarity and size. Also, these methods focus on patterns that map substructures. However, during testing, the evaluation metrics for molecule quality are often based on the entire molecule. These discrepancies cause two challenges. We also empirically demonstrate them using Junction Tree Variational Auto-Encoder (JTVAE) [11] and Variational Junction Tree encoder-decoder (VJTNN) [18] models, as shown in Table Table 4.1.

- **Difficulty in maintaining similarity while optimizing drug properties**. Similarity scores between input and generated molecules are relatively low compared with property improvement. For example, when property scores are higher than 0.7 in the range $[0, 1]$, the similarity score will only be low (around 0.3 in the range $[0, 1]$), as shown in Table Table 4.1.

- **Difficulty in maintaining molecule sizes which affect property improvement**. When generated molecules are of very different sizes compared to the target molecules, they will perform poorly in both similarity and property. Table Table 4.1 show the property improvement as a function of the size difference between the generated and input molecules. We can see that a large size difference in either positive or negative direction usually corresponds to smaller property improvement.

Dataset	Method	All Test Set		Small Molecule		Large Molecule	
		Similarity	Property	Similarity	Property	Similarity	Property
QED	JTVAE [11]	0.298	0.804	0.142	0.542	0.231	0.792
	VJTNN [18]	0.311	0.885	0.178	0.602	0.283	0.818
DRD2	JTVAE	0.299	0.709	0.164	0.501	0.243	0.621
	VJTNN	0.315	0.765	0.170	0.579	0.296	0.744

Table 4.1: Empirical results of two deep generative models on two molecule optimization tasks.

4.2 Main Idea and Contribution

To address these challenges, we introduce Molecule-level Rewards (MOLER) to enhance the molecule-level properties. Graph-to-graph model is the base model for MOLER (MOLER enhance graph-to-graph models), graph-to-graph model has been described in chapter 3. MOLER is a general and flexible approach that can be incorporated into various deep generative models for improved performance. MOLER is enabled by the following technical contributions.

- **Similarity reward** (Section subsection 4.3.1): We formulate the similarity between input and generated molecules as a reward function to alleviate the **similarity gap** between them.
- **Size deviation penalty** (Section subsection 4.3.2): To explicitly control the size of the molecule, we design a size deviation penalty that penalizes large size deviations to reduce the **size difference** between the target and generated molecules.

4.3 MOLER Framework

In this section, we describe Molecule-Level Reward (MOLER). For ease of exposition, we list all the mathematical notations in Table Table 4.2.

Table 4.2: Notations used in MOLER.

Notations	Explanations
(X, Y)	input-target molecule pair
S	substructure set S (a.k.a. vocabulary)
V/E	set of vertex(atom) / edge(bond)
$G = (V, E)$	molecular graph
$\mathcal{T}_G = (\mathcal{V}, \mathcal{E})$	scaffolding tree of graph G, junction tree [11]
$\mathbb{T}_G$	scaffolding tree of G without substructure, output of topological prediction
$N(v)$	set of neighbor nodes of vertex v
$\mathbf{e}_v/\mathbf{e}_{uv}$	feature vector for node v / edge (u, v)
$\mathbf{m}_{uv}^{(t)}$	message for edge (u, v) at the t-th iteration
T	Depth of Message Passing Network
$\mathbf{z}_i^G/\mathbf{z}_i^{\mathcal{T}}$	embedding of node i in G / $\mathcal{T}$, $\mathbf{Z}^G = \{\mathbf{z}_1^G, \cdots\}$
$\mathbf{h}_{i_t,j_t}$	message vector for edge (i_t, j_t)
$\mathbf{d}_G$	Embedding of Graph G
p_t^{topo}	topological prediction score at the t-th step
$\mathbf{q}_t^{\text{sub}}$	substructure prediction distribution at the t-th step
$f_i(\cdot),\ i = 1, \cdots, 6$	parameterized neural networks
$\text{sim}(X, Y) \in [0, 1]$	Similarity between X and Y
$\text{size}(X)$	number of substructure in X
θ	all learnable parameters in generative model
$\pi_\theta(Y\|X)$	generative model parameterized by θ
$g_1(\cdot), g_2(\cdot)$	fully-connected neural network

4.3.1 Similarity Reward

Both similarity with input molecule X and property of Y (denoted $\text{sim}(X, Y)$) are essential metrics to evaluate the quality of generated Y. We consider adding "similarity reward" to explicitly enhance the similarity constraint, i.e., maximize

$$\mathcal{L}_{\text{sim}}(\theta) = \mathbb{E}_{Y \sim \pi_\theta(\cdot|X)}\big[\text{sim}(X, Y)\big], \tag{4.1}$$

where θ represents all the parameters for generative models; $\mathcal{L}_{\text{sim}}(\theta)$ is objective function for similarity reward; hyperparameter $w_{\text{sim}} \in \mathbb{R}_+$ is weight of the reward. $\text{sim}(X, Y)$ represents Tanimoto similarity between molecule X and Y. When optimizing Equation (Equation 4.1), the computation of its gradient estimator requires the numerical value of the probability density function $\pi_\theta(Y|X)$ explicitly. Now we discuss how to evaluate $\pi_\theta(Y|X)$ explicitly.

We know from chapter 3, the molecule generation is mainly divided into three prediction tasks: (i) topological prediction, which is a binary classification task; (ii) substructure prediction; (iii)Assembling prediction (in graph decoder). Since there are two phases (scaffolding tree and molecular graph) in graph generation, $\pi_\theta(Y|X)$ can be written as the following joint distribution that incorporate the generation of both the scaffolding tree and the molecular graph, which includes three prediction tasks,

$$\pi_\theta(Y|X) = \underbrace{\prod_{t=1}^{2|\mathcal{E}|} p_t^{\text{topo}}}_{\text{topological prediction}} \cdot \underbrace{\prod_{t \in \mathcal{S}} \mathbf{q}_t^{\text{sub}}}_{\text{substructure prediction}} \cdot \underbrace{\frac{\exp(s^a(G_Y))}{\sum_{G_i} \exp(s^a(G_i))}}_{\text{assembling prediction}}. \tag{4.2}$$

First, p_t^{topo} is the probability for the t-th topological prediction. $\mathcal{E}$ is edge set of the generated scaffolding tree. Since the generation procedure would terminate until backtracking to root node, each edge is visited twice and there are totally $2|\mathcal{E}|$ topological predictions. Second, regarding substructure prediction, $\mathbf{q}_t^{\text{sub}}$ is a distribution over all substructures, $\mathcal{S} \subseteq \{1, 2, \cdots, 2|\mathcal{E}|\}$ represents the set of the indexes of the edges who expands to a new node in a scaffolding tree, because only when topological prediction p_t^{topo} predict to expand to a new node, we need to make substructure prediction.

4.3.2 Size Deviation Penalty

Size deviation between input and target molecules is another reason that the target molecule is dissimilar to the input. Therefore, we design a penalty score to constrain this deviation.

In graph generation, there are three key prediction tasks: (i) topological prediction; (ii)

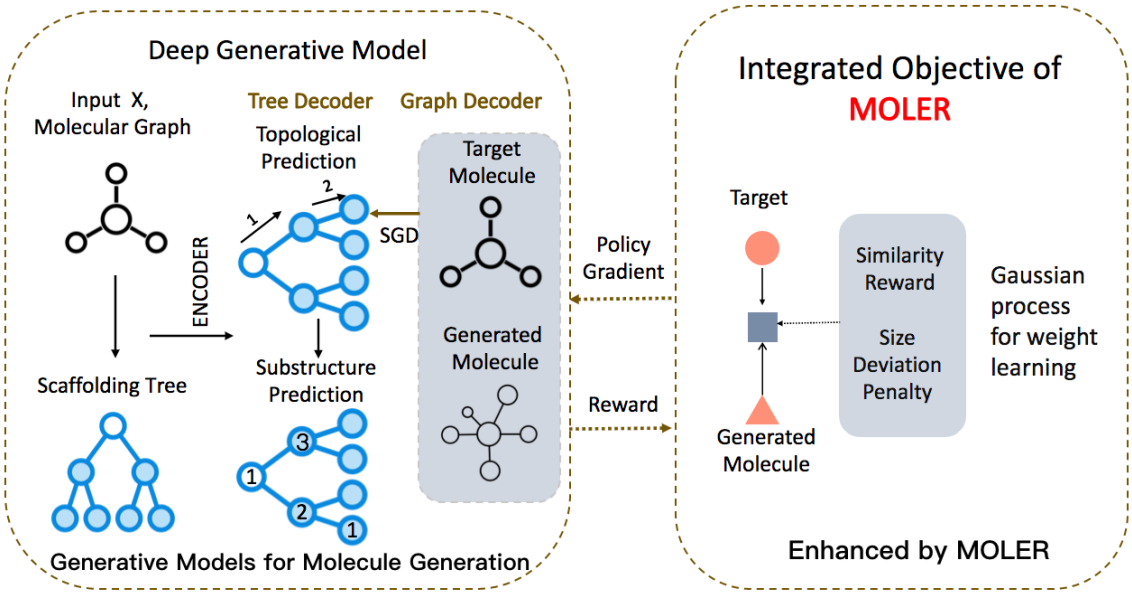


Figure 4.1: MOLER Framework.

substructure prediction; (iii) assembling prediction. Since scaffolding tree-based approaches use substructure as the basic component in molecule generation, we use the number of substructures as the surrogate for molecule size, which is much more efficient to compute since it is only related to topological prediction.

To design a reasonable size deviation penalty, we empirically investigate the correlation between the size of X and Y in training data pairs and find they are positively correlated. We want to minimize the following size deviation penalty,

$$\mathcal{L}_{\text{size}}(\theta) = \mathbb{E}_{\pi_\theta^{\mathbb{T}}(\mathbb{T}_Y|X)}\Big[g\big(\text{size}(\mathbb{T}_X), \text{size}(\mathbb{T}_Y)\big)\Big], \tag{4.3}$$

where $\mathcal{L}_{\text{size}}(\theta)$ is objective function of size deviation penalty, hyperparameter $w_{\text{size}} \in \mathbb{R}_+$ is weight for size deviation penalty, $\text{size}(X)$ denotes the number of substructure in scaffolding tree of X. $g(\cdot,\cdot)$ is the reward function of molecule size defined as

$$g(x,y) = \begin{cases} |x-y| - \epsilon, & |x-y| > \epsilon, \\ 0, & |x-y| \leq \epsilon, \end{cases} \tag{4.4}$$

Algorithm 1 MOLER

# training
while Convergence criteria is not met **do**
    Sample a minibatch $\mathcal{M} = \{(X_1, Y_1), \cdots, (X_m, Y_m)\}$, $m = |\mathcal{M}|$.
    Optimize $\mathcal{L}_{\text{gen}}(\theta)$ w.r.t. θ using $\{(X_i, Y_i)\}_{i=1}^m$ using SGD, where $\mathcal{L}_{\text{gen}}(\theta)$ is loss of generative models.
    Generate molecule via $\tilde{Y}_i \sim \pi_\theta(\cdot|X_i)$ for $i = 1, \cdots, m$.
    Maximize $\mathcal{L}_{\text{sim}}(\theta) + \mathcal{L}_{\text{size}}(\theta)$ w.r.t. θ using $\{(X_i, \tilde{Y}_i)\}_{i=1}^m$ based on policy gradient.
end while
# test, e.g., QED task
for $X_i \in$ Test Set **do**
    generate $Y_i \sim \pi_\theta(\cdot|X_i)$.
    Evaluate and record $\text{sim}(X_i, Y_i)$ and $\text{QED}(Y) - \text{QED}(X)$
end for
Evaluate average similarity, property improvement, and success rate on the whole test set.

where $\epsilon \in \mathbb{N}_+$ is a positive integer. Then we discuss the optimization procedure. Without loss of generalization, we consider maximizing a general objective as follows,

$$\mathcal{L}(\theta) = \mathbb{E}_{Y \sim \pi_\theta(\cdot|X)}\big[R(X, Y)\big], \tag{4.5}$$

where $\mathcal{L}(\theta)$ can be either $\mathcal{L}_{\text{sim}}$ or $\mathcal{L}_{\text{size}}$. $\nabla_\theta \mathcal{L}(\theta)$, the gradient of objective function with regard to θ, can be simplified via policy gradient,

$$\nabla_\theta \mathcal{L}(\theta) = \mathbb{E}_{Y \sim \pi_\theta(\cdot|X)}\Big[\nabla_\theta \log \pi_\theta(Y|X) R(X, Y)\Big]. \tag{4.6}$$

4.3.3 Choosing Weight for MOLER

During the learning procedure, two reward values (related to similarity and size) are integrated into the learning objective in order to minimize

$$\mathcal{L} = \mathcal{L}_{\text{gen}} - w_{\text{sim}} \mathcal{L}_{\text{sim}} + w_{\text{size}} \mathcal{L}_{\text{size}}, \tag{4.7}$$

where $w_{\text{sim}}, w_{\text{size}} \in \mathbb{R}_+$ are hyperparameters that control the strength of MOLER, $\mathcal{L}_{\text{gen}}$ is loss of generative model . It is time-consuming to use grid search/random search to find the near-optimal weight combination $(w_{\text{sim}}, w_{\text{size}})$. To address this issue, we resort to the Gaussian process (GP) [51]. The gaussian process is used to approximate the validation accuracy as a function of these weight combinations. The validation accuracy can be the success rate, which would be described later). Then via searching for the optimum of the approximated function, we obtain the appropriate weight combinations. Concretely, we denote the function to approximate as $h(w_1, w_2)$, where $w_1 = w_{\text{sim}}, w_2 = w_{\text{size}}$. The search range for $w_i,\ i = 1, 2$ is bounded by $\text{LB}_i \leq w_i \leq \text{UB}_i$. We draw m weight combinations, denoted $\mathbf{w}^{(1)} = (w_1^1, w_2^1); \mathbf{w}^{(2)} = (w_1^2, w_2^2); \cdots; \mathbf{w}^{(m)} = (w_1^m, w_2^m)$. For the i-th weight combination $\mathbf{w}^{(i)}$, we evaluate the task metric (e.g., success rate) on the validation set r^i, which can be seen as a noisy evaluation of the true function $h(w_1^i, w_2^i)$ we want to approximate. The noise comes from the sampling bias of the validation set. The true objective function $h(w_1, w_2)$ is unknown, a zero-mean Gaussian prior is specified,

$$h(\cdot) \sim \text{GP}(\cdot, k(\cdot, \cdot)), \tag{4.8}$$

where $k(\cdot, \cdot)$ is the covariance function. Given m points $\{\mathbf{w}^{(i)}\}_{i=1}^m$ (weight combination) and their evaluation $\{r^i\}_{i=1}^m$. We assume r is a noisy evaluation of the true unknown function that we are interested, i.e., $r(\mathbf{w}) = h(\mathbf{w}) + \epsilon,\ \mathbf{w} = (w_1, w_2)$. According to Bayes rule, the posterior distribution of the true objective function is

$$\begin{aligned} h|\{\mathbf{w}^{(i)}, r^i\}_{i=1}^m &\sim \mathcal{N}(\mu(\mathbf{w}), \sigma^2(\mathbf{w})), \\ \mu(\mathbf{w}) &= \mathbf{k}^\top (\mathbf{K} + \sigma^2 I)^{-1} r, \\ \sigma(\mathbf{w}) &= k(\mathbf{w}, \mathbf{w}) - \mathbf{k}^\top (\mathbf{K} + \sigma_n^2 I)^{-1} k, \end{aligned} \tag{4.9}$$

where

$$\mathbf{K} = \begin{bmatrix} k(\mathbf{w}_1, \mathbf{w}_1) & \cdots & k(\mathbf{w}_1, \mathbf{w}_m) \\ \vdots & \ddots & \vdots \\ k(\mathbf{w}_m, \mathbf{w}_1) & \cdots & k(\mathbf{w}_m, \mathbf{w}_m) \end{bmatrix},$$
$$\mathbf{k} = [k(\mathbf{w}, \mathbf{w}_1), \cdots, k(\mathbf{w}, \mathbf{w}_m)]^\top, \tag{4.10}$$
$$r = [r_1, \cdots, r_m]^\top.$$

Regarding covariance function we set $k(\mathbf{w}, \mathbf{v}) = \exp\big(-\frac{1}{2}(\mathbf{w}-\mathbf{v})^\top \Sigma^{-1}(\mathbf{w}-\mathbf{v})\big)$, where $\Sigma = \text{diag}\big(\alpha(\text{UB}_1 - \text{LB}_1)^2, \alpha(\text{UB}_2 - \text{LB}_2)^2\big)$. α is empirically set to 0.2 [24].

To summarize, the Gaussian process provides a surrogate function to approximate the true objective $h(w_1, w_2)$, the validation accuracy as a function of weights in MOLER (w_{sim} and w_{size}). The surrogate can be used to search efficiently for the optimum of the objective function, i.e., optimal weights in MOLER. Grid search is leveraged here to explore the surrogate and get the optimal weights.

4.4 Main results

The experimental setup follows CORE's setup with the same dataset, properties, evaluation metrics, baseline methods (including CORE), etc. Then we demonstrate that MOLER can improve the state-of-the-art generative model methods on all three datasets. The results (in terms of similarity, property improvement, and success rate) are reported in Table Table 4.3. We found that MOLER variants (JTVAE+MOLER, VJTNN+MOLER, CORE+MOLER) provide significant and consistent improvement across different generative models up to 19% compared to their base model (JTVAE, VJTNN, CORE).

The positive effect of similarity reward. Next, we present the effect of similarity reward defined in Equation (Equation 4.1) (Section subsection 4.3.1) and show how we set the similarity reward. In practice, to accelerate the optimization procedure [53], we want to make the average $R(X, Y)$ to be close to 0, so we subtract its average from the original

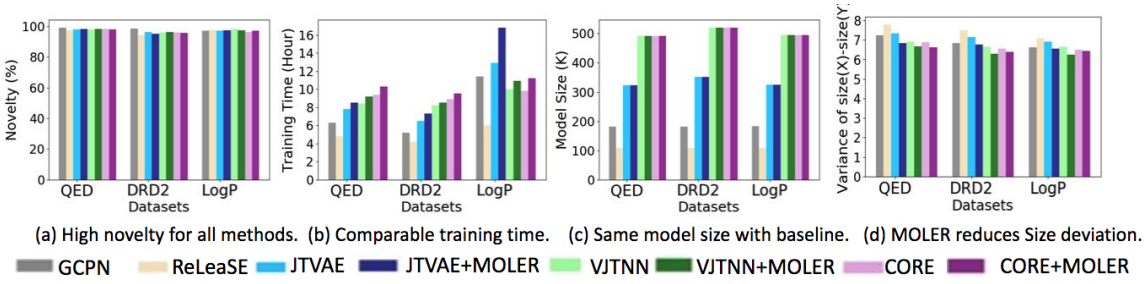


Figure 4.2: Results measured by novelty, training time, model size, and size deviation of the generated molecule. (a) All methods have $\sim 100\%$ novelty scores, and (b-c) MOLER variants do not cost much more computationally cost in training time, and their model sizes are compared to the original generative models. (d) MOLER can reduce the size deviation of generated molecules.

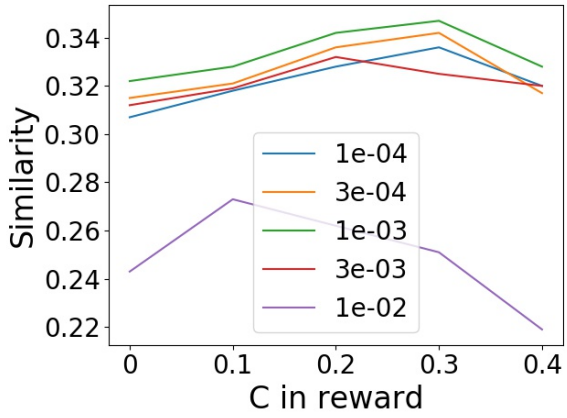


Figure 4.3: Selection of (w_{sim}, C) on QED dataset for similarity reward (Section subsection 4.3.1). We find (i) $w_{\text{sim}} = 1e^{-3}$ and $C = 0.3$ performs best among all the hyper-parameters, especially on the improvement of similarity; (ii) within a reasonable range, the similarity would increase when weight w_{sim} increase. However, too large w_{sim} would degrade the performance even worse than the baseline method (VJTNN, dashed line).

Table 4.3: Experimental results of MOLER. MOLER-based methods outperform deep generative methods (JTVAE [11], VJTNN [18] and CORE [1]) and RL methods (GCPN [26], ReLeaSE [52]). MOLER-based methods achieved the best performance in all settings, especially CORE+MOLER, which seems the most competitive. MOLER provides significant and consistent improvement across different generative models up to 19%. Green text highlights the ones with improvement of $> 5\%$. In each column, we highlight the best performance using bold font.

Method	Similarity			Property Improvement			Success Rate (%)		
	QED	DRD2	LogP	QED	DRD2	LogP	QED	DRD2	LogP
GCPN	0.308	0.309	0.360	0.877	0.745	3.041	47.71	44.81	55.43
JTVAE	0.299	0.300	0.285	0.791	0.693	2.532	38.74	35.43	38.43
JTVAE+MOLER	0.302	0.314	0.315	0.858	0.732	3.015	43.20	40.01	45.24
improvement	+1.07%	+4.77%	+10.41%	+8.47%	+5.63%	+19.08%	+11.51%	+12.93%	+17.72%
VJTNN	0.315	0.316	0.358	0.886	0.764	2.972	48.16	45.38	55.15
VJTNN+MOLER	0.351	0.334	**0.372**	0.904	0.778	3.182	56.32	47.39	57.01
improvement	+11.61%	+5.56%	+3.94%	+2.03%	+1.83%	+7.07%	+16.94%	+4.43%	+3.37%
CORE	0.321	0.333	0.369	0.895	0.769	3.100	50.26	47.91	57.64
CORE+MOLER	**0.360**	**0.352**	0.371	**0.910**	**0.782**	**3.199**	**57.32**	**49.47**	**57.93**
improvement	+12.11%	+5.58%	+.41%	+1.68%	+1.69%	+3.19%	+14.05%	+3.26%	+.50%

value. For example, in similarity reward, the reward is

$$R(X, Y) = \text{sim}(X, Y) - C, \tag{4.11}$$

where $C \in \mathbb{R}$ is a hyperparameter, we want it to be close to the average of all the similarity values.

Two hyperparameters play a crucial role in the empirical performance of similarity reward: (1) the weight of similarity reward in the whole objective $w_{\text{sim}} \in \mathbb{R}_+$ in Equation (Equation 4.1); (2) hyperparameter in similarity reward $C \in \mathbb{R}$ in Equation (Equation 4.11). We search the weight of similarity reward w_{sim} from $\{1e^{-4}, 3e^{-4}, 1e^{-3}, 3e^{-3}, 1e^{-2}\}$. For the hyperparameter in similarity reward C, we want it to be close to the average of the similarity value. During the learning procedure, the similarity would increase from 0 to about 0.3-0.4. So we search C from $\{0, 0.1, 0.2, 0.3, 0.4\}$. We use grid search to find the optimal combination of hyperparameters. $w_{\text{sim}} = 0$ corresponds to the baseline method (VJTNN) that does not use similarity reward. We can find that most of the (w_{sim}, C) combinations can outperform the baseline method, validating the effectiveness of adding similarity reward.

In addition, we show more visualization results in Figure Figure 4.3. For each weight $w_{\text{sim}} \in \{1e^{-4}, 3e^{-4}, 1e^{-3}, 3e^{-3}, 1e^{-2}\}$, we show the change of performance (both similarity and property) with different Cs (in Equation (Equation 4.11)) in Figure Figure 4.3(a) and (b). Also, for each $C \in \{0, 0.1, 0.2, 0.3, 0.4\}$, we show the change of performance with various w_{sim} in Figure Figure 4.3 (c) and (d). We find that (i) $w_{\text{sim}} = 1e^{-3}$ performs best among all the hyperparameters, especially on the improvement of similarity; (ii) When $C = 0.3$, MOLER achieve the best performance; (iii) within a reasonable range, the similarity would increase as we increase the weight w_{sim}; (iv) too large w_{sim} would degrade the performance even worse than the baseline method (VJTNN).

For QED dataset, the optimal combination of hyperparameters is $w_{\text{sim}} = 1e^{-3}$ and $C = 0.3$, as mentioned above. It is also validated to be optimal in the DRD2 dataset. For LogP dataset, the optimum is $w_{\text{sim}} = 1e^{-3}$ and $C = 0.4$.

For each size deviation penalty g, we plot the change of performance (both similarity and property) with various weight w_{size} Figure Figure 4.3 on the QED dataset. We find that (i) g_5 (which is represented in Section subsection 4.3.2) and $w_{\text{size}} = 1e^{-3}$ achieve the best performance; (ii) most of the hyperparameter setting would outperform VJTNN (baseline). Therefore, the effect of the size deviation penalty is positive [19]. Moreover, this selection is also validated to be optimal on both DRD2 and LogP datasets.

4.5 Conclusion and Discussion

In this work, we have proposed to incorporate molecule level reward function (MOLER) into deep generative models for molecule optimization. Specifically, we have designed two molecule reward functions motivated by some empirical observations. The first one is the similarity reward to encourage the generated molecule to be similar to the input one. Another reward is to control the size of the generated molecule explicitly. MOLER provides a general and flexible framework that can incorporate various reward functions to specify different aspects of generated molecules based on any deep generative models for

molecule optimization. Policy gradient is applied to optimize the reward objective, and it wouldn't cause too much extra computational cost compared with deep generative models. Thorough empirical studies have been conducted on several real molecule optimization tasks to validate the effectiveness of MOLER.

Part II

Self-supervised learning

Overview In the second part of the thesis, we focus on leveraging the self-supervised learning method for drug design methods. Self-Supervised Learning (SSL) is a machine learning paradigm where a model learns from unlabeled data and generates data labels automatically from the unlabeled data, and the labels are further used in subsequent iterations as ground truths. There are two kinds of SSL paradigms: generative SSL and contrastive SSL. Specifically, generative SSL masks a subset of structured data and trains a machine learning model (mostly neural networks) to predict the masked part based on the remaining data while contrastive SSL builds a positive and negative view of structured data via mutating the structured data and then train a machine learning model (mostly neural networks) to discriminate the negative data from the positive one. In this thesis, we restrict our attention to generative SSL. The main advantage of generative SSL is that it can construct the conditional distribution of the structured data via masking a subset of raw data and predicting its category conditioned on the rest of the raw data. Also, it is learned on massive unlabeled data and does not require too many labeled data, thus being data-efficient. Most existing self-supervised learning methods focus on learning a representation to serve as a warm start for downstream applications. In contrast, we leverage the conditional distribution learned by self-supervised learning for drug molecule generation. Specifically, on small-molecule drug design, we proposed *Multi-constraint Molecule Sampling (MIMOSA)* that formulates molecule optimization as a sampling problem. Then in each sampling step, we sample from the conditional distribution, a graph neural network pretrained on a large unlabeled chemical compound library with self-supervised learning.

> chapter 5: **MIMOSA: Multi-constraint Molecule Sampling for Molecule Optimization.** Tianfan Fu, Cao Xiao, Xinhao Li, Lucas M. Glass, Jimeng Sun. Association for the Advancement of Artificial Intelligence (AAAI) 2021.

Then, we also formulate the inverse protein folding task (also known as fixed-backbone protein design, which is a fundamental problem in biologics design) into a sampling problem and use self-supervised learning to build a conditional probability which we can sample

from. Compared with existing methods (mostly based on maximum likelihood learning and autoregressive models) that sequentially generate amino acids, it provides more flexibility that adaptively updates the amino acids with uncertainty quantification.

chapter 6: **SIPF: Sampling Method for Inverse Protein Folding.** Tianfan Fu, Jimeng Sun. The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (2022).

CHAPTER 5
MIMOSA: MULTI-CONSTRAINT MOLECULE SAMPLING FOR MOLECULE OPTIMIZATION

5.1 Research Challenge

Existing works on molecule optimization and molecule generation tasks can be categorized as generative models [54, 55, 10] and reinforcement learning (RL) methods [26, 27, 19, 56]. Most existing works only optimize a single property, while multiple properties must be optimized to develop viable drug candidates. Recently, [28] proposed a molecule generation algorithm that can optimize multiple properties. It is a related but different task than molecule optimization since they do not take any specific input molecule as the anchor. However, this method is restricted by the assumption that each property is associated with specific molecular substructures, which does not apply to global properties like logP. [29] proposed a genetic algorithm (GA) for molecule generation and optimization. However, the model is likely to get trapped in regions of local optima, thus the performance would degrade in optimizing properties that are less sensitive to the change of local structures such as molecule's biological activity against a dopamine type 2 receptor (DRD) [29].

5.2 Main Ideas and Contributions

To allow for flexible and efficient molecule optimization on multiple properties, we propose a new sampling-based molecule optimization framework named MultI-constraint MOlecule SAmpling (MIMOSA). The main contributions are:

- **Formulation**. A new sampling framework for flexible encoding of multiple constraints. We reformulate the molecule optimization task in a sampling framework to draw molecules from the target distribution. The framework provides flexible and efficient encoding of

Notations	short explanation
X, Y	Input molecule, target molecule.
$\text{sim}(X, Y) \in [0, 1]$	Similarity of molecules X and Y.
$p_X(Y)$	Target dist. when optimizing X, Equation Equation 5.1.
M	# of properties to optimize.
$\gamma_0, \gamma_1, \cdots, \gamma_M \in \mathbb{R}_+$	Hyperparameter in Target dist. $p_X(Y)$.
$\mathcal{P}_1, \cdots, \mathcal{P}_M$	Molecular properties to optimize.
$\mathbf{1}(Y)$	Validity Indicator func. of molecule Y.
K	Depth of GNN.
$\mathbf{h}_v^{(k)} \in \mathbb{R}^{300}$	Node embedding v in the k-th layer.
C_1/C_2	# of all possible substructures/bonds.
$v; s_v/s'_v$	node v; substructures of v.
$\mathbf{f}_v/\mathbf{g}_e$	one-hot node/edge feature.
$\hat{\mathbf{y}}_v/\text{mGNN}(Y, v)$	substructure distribution. Equation (Equation 5.2).
$\hat{z}_v$/bGNN(Y, v)	probability of v will expand. Equation (Equation 5.3).
$\mathbf{y}_v/z_v$	ground truth label of node v
mGNN(Y,v)	substructure prediction on node v
bGNN(Y,v)	prediction whether to extend on node v
Y/Y'	current/next Sample.
$S_{\text{add}}, S_{\text{replace}}, S_{\text{delete}}$	sampling operation from Y to Y'.

Table 5.1: Notations used in MIMOSA.

multi-property and similarity constraints (Section subsection 5.3.1).

- **Method**. Efficient sampling augmented by GNN pretraining. With the help of two pretrained GNN models, we designed a molecule sampling method that enables efficient sampling from a target distribution. This enables MIMOSA to leverage the vast amount of molecule data in a self-supervised manner without the need for labeled data (Section subsection 5.3.2).

- **Experiment**. We compare MIMOSA with state-of-the-art baselines on optimizing several important properties across multiple settings, MIMOSA achieves 43.7% success rate (49.1% relative improvement over the best baseline GA [29]) (Section section 5.4).

5.3 MIMOSA Framework

In this section, we describe Multi-constraint Molecule Sampling for Molecule Optimization (MIMOSA). For ease of exposition, we list all the mathematical notations.

5.3.1 Molecule Optimization via Sampling

We formulate molecule optimization into a sampling problem. Here to formulate molecule optimization that aims to optimize on the similarity between the input molecule X and the target molecules Y as well as M molecular properties of Y, $\mathcal{P}_1, \cdots, \mathcal{P}_M$ (the higher score, the better). We propose to draw Y from the *unnormalized target distribution*:

$$p_X(Y) \propto \exp\left(\eta_0 \text{sim}(X, Y) + \eta_1 \sum_{i=1}^{M} \mathcal{P}_i(Y) - \mathcal{P}_i(X)\right), \tag{5.1}$$

where $\eta_0, \eta_1, \cdots, \eta_M \in \mathbb{R}_+$ are the hyperparameters. The target distribution encodes similarity and multiple property constraints. $\text{sim}(X, Y)$ is similarity between X and Y.

5.3.2 The MIMOSA Method for Molecule Sampling

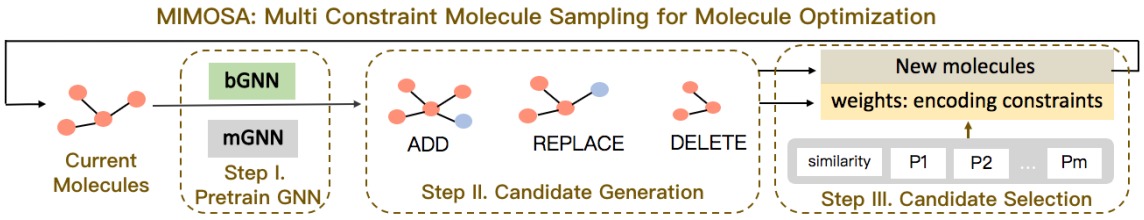


Figure 5.1: MIMOSA framework.

Fig. Figure 5.1 illustrates the overall procedure of MIMOSA, which can be decomposed into the following steps: (1) Pretrain GNN. MIMOSA pre-trains two graph neural networks (GNNs) using a large number of unlabeled molecules, which will be used in the sampling process. Then MIMOSA iterates over the following two steps. (2) Candidate Generation. We generate and score molecule candidates via modification operations (*add, delete, replace*) to the current molecule. (3) Candidate Selection. We perform MCMC sampling to select promising molecule candidates for the next sampling iteration by repeating Step 1-3.

(I) Pretrain GNNs for Substructure-type and Molecule Topology Prediction

We develop two GNN-based pretraining tasks to assist molecule modification: mGNN and bGNN.

(1) **mGNN** model aims at multi-class classification for predicting the substructure type of a masked node. We mask the individual substructure, and replace it with a special masked indicator following [57]. Suppose we only mask one substructure for each molecule during training and v is the masked substructure (i.e., node), y_v is the node label corresponding to masked substructure type, we add fully-connected (FC) layers with softmax activation to predict the type of node v: $\hat{\mathbf{y}}_v = \mathrm{Softmax}\big(\mathrm{FC}(\mathbf{h}_v^{(K)})\big)$. where $\hat{\mathbf{y}}_\mathbf{v}$ is a C_1 dimension vector, indicating the predicted probability of all possible substructures. Multi-class cross-entropy loss is used to guide GNN training. To summarize, the prediction of mGNN is defined as

$$\hat{\mathbf{y}}_v \triangleq \text{mGNN}(Y, \text{mask} = v) = \text{mGNN}(Y, v), \tag{5.2}$$

where in a given molecule Y the node v is masked, mGNN predicts the substructure distribution on masked node v, which is denoted $\hat{\mathbf{y}}_v$.

(2) **bGNN** is designed to predict whether a node will expand. To provide training labels for bGNN, we set the leaf nodes (nodes with degree 1) with label $z_v = 0$ as we assume they are no longer expanding. And we set label $z_v = 1$ on the non-leaf nodes that are adjacent to leaf nodes as those nodes expanded (to the leaf nodes). The prediction is done via $\hat{z}_v = \mathrm{Sigmoid}\big(\mathrm{FC}(\mathbf{h}_v^{(K)})\big)$, where FC is two-layer fully-connected layers. $\mathbf{h}_v^{(K)}$ is the node embedding of v produced by GNN. In sum, the prediction of bGNN is defined as

$$\hat{z}_v \triangleq \text{bGNN}(Y, v), \tag{5.3}$$

where v is a node in molecule Y, $\hat{z}_v$ is the probability that v will expand.

(II) Candidate Generation via Substructure Modification Operation

With the help of mGNN and bGNN, we define substructure modification operations namely *replace, add or delete* on input molecule Y: That is, each time we replace, add or delete one substructure based on the given molecule. Let Y be the current molecule graph, and Y' be the generated molecule graph after substructure modification operations. We consider single iteration of the MCMC method. Specifically, we suppose Y is the current molecule, and G_Y is the corresponding molecular graph.

- Replace a substructure. At node v, the original substructure category is s_v. It has three key steps: (1) We mask v in Y, evaluate the substructure distribution in v via mGNN, i.e., $\hat{\mathbf{y}}_v = \text{mGNN}(Y, v)$, as Equation (Equation 5.2). (2) Then we sample a new substructure s'_v from the multinomial distribution $\hat{\mathbf{y}}_v$, denoted by $s'_v \sim \text{Multinomial}(\hat{\mathbf{y}}_v)$. (3) At node v, we replace the original substructure s_v with new substructure s'_v to produce the new molecule Y'. The whole operation is denoted as

$$Y' \sim S_{\text{replace}}(Y'|Y). \tag{5.4}$$

- Add a substructure. Suppose we want to add a substructure as leaf node (denoted as v) connecting to an existing node u in the current molecule Y. The substructure category of v is denoted s_v, which we want to predict. It contains 3 key steps: (1) We evaluate the probability that node u has a leaf node v with the help of bGNN in Equation (Equation 5.3), i.e., $\hat{z}_u = \text{bGNN}(Y, u) \in [0, 1]$. (2) Suppose the above prediction is to add a leaf node v. We then generate a new molecule Y' via adding v to Y via a new edge (u, v). (3) In Y', s_v, the substructure of v is unknown. We will predict its substructure using mGNN, i.e., $\hat{\mathbf{y}}_v = \text{mGNN}(Y', v)$, following Equation (Equation 5.2). (4) sample a new substructure s'_v from the multinomial distribution $\hat{\mathbf{y}}_v$ and complete the new molecule Y'. The whole operation is denoted as

$$Y' \sim S_{\text{add}}(Y'|Y). \tag{5.5}$$

• Delete a substructure. We delete a leaf node v in the current molecule Y. It is denoted

$$Y' \sim S_{\text{delete}}(Y'|Y). \tag{5.6}$$

In the MCMC process, $S_*(Y'|Y)$ indicates the sequential sampling process from the previous sample Y to the next sample Y'. And the very first sample is the input X.

(III) Candidate Selection via MCMC Sampling

The set of generated candidate molecules can be grouped as three sets based on the type of substructure modification they received, namely, *replace* set S_{replace}, *add* set S_{add}, and *delete* set S_{delete}.

Sampling S_{replace}. For molecules produced by the "replace" operation, the weight in sampling w_r is

$$w_r = \frac{p_X(Y') \cdot [\text{mGNN}(Y, v)]_{s'_v}}{p_X(Y) \cdot [\text{mGNN}(Y, v)]_{s_v}}, \tag{5.7}$$

where $P_X(\cdot)$ is the unnormalized target distribution for optimizing X, defined in Equation (Equation 5.1), $[\text{mGNN}(Y, v)]_{s_v}$ is the predicted probability of the substructure s_v in the prediction distribution $\text{mGNN}(Y, v)$. The acceptance rate in the proposal is $\min\{1, w_r\}$. If the proposal is accepted, we use the new prediction s'_v to replace the origin substructure s_v in the current molecule Y and produce the new molecule Y'.

Sampling S_{add}. For molecules produced by the "add" operation, the weight in sampling is

$$w_a = \frac{p_X(Y') \cdot \text{bGNN}(Y, u) \cdot [\text{mGNN}(Y', v)]_{s_v}}{p_X(Y) \cdot (1 - \text{bGNN}(Y, u))}, \tag{5.8}$$

where acceptance rate in the proposal is $\min\{1, w_a\}$.

5.4 Main Results

We first describe the experimental setup. We use 250K molecules from ZINC database [45] to train both mGNN and bGNN. We focus on the 3 molecular properties, as described in chapter 3. We consider the following baselines: JTVAE, VJTNN, GCPN as mentioned in CORE (chapter 3). We also incorporate GA (Genetic Algorithm) [29].

Evaluation Strategies. To optimize "Penalized LogP", we randomly select 500 molecules from ZINC database [45]. To optimize "DRD" and "DRD & Penalized LogP", following [18] we pick 500 molecules whose DRD scores are lower than 0.05. For the "QED" and "QED & Penalized LogP" tasks, following [18] we pick 500 molecules with QED scores $[0.7, 0.8]$. Data in validation and test are not in training.

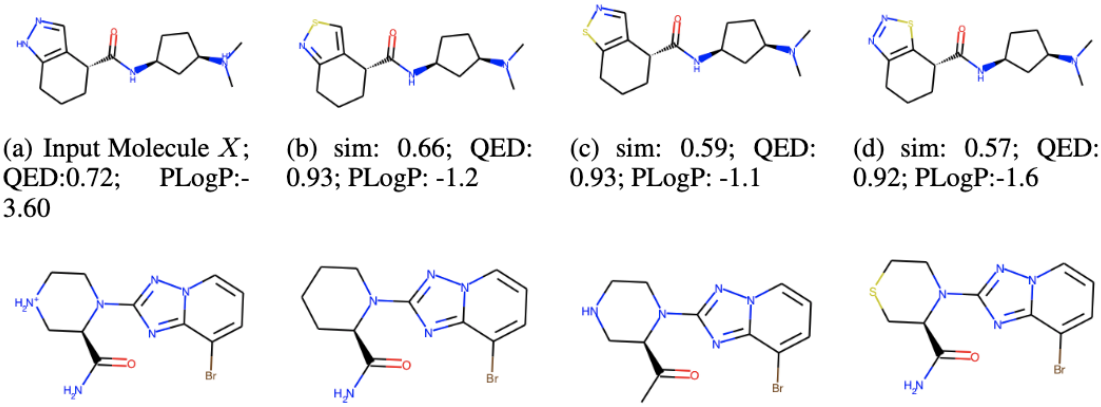
Optimizing PLogP and QED				
Method	Similarity	PLogP-Imp.	QED-Imp.	Success
JTVAE	0.16±0.08	0.14±0.27	0.01±0.10	0.4%
VJTNN	0.17±0.06	0.46±0.35	0.02±0.09	1.0%
GCPN	0.25±0.15	0.56±0.25	0.06±0.08	11.3%
SELFIES-GA+D	0.35±0.16	**0.93±0.67**	0.09±0.07	24.9%
MIMOSA	**0.42±0.17**	**0.93±0.48**	**0.10±0.09**	**32.0%**
Optimizing PLogP and DRD				
Method	Similarity	PLogP-Imp.	DRD-Imp.	Success
JTVAE	0.18±0.08	0.20±0.18	0.18±0.09	0.8%
VJTNN	0.18±0.08	0.55±0.16	0.27±0.05	3.4%
GCPN	0.23±0.12	0.38±0.25	0.25±0.11	20.4%
SELFIES-GA+D	0.38±0.16	0.68±0.49	0.20±0.16	29.3%
MIMOSA	**0.54±0.16**	**0.75±0.48**	**0.35±0.20**	**43.7%**

Table 5.2: Multi-properties optimization.

To evaluate model performance in optimizing multiple drug properties, we consider the following combinations of property constraints: (1) optimize QED and PLogP; (2) optimize DRD and PLogP. From Table Table 5.2, MIMOSA has significantly better and stable performance on all metrics, with 28.5% relative higher success rate in optimizing both QED and PLogP, and 49.1% relative higher success rate in optimizing both DRD and PLogP compared with the second best algorithm GA. The GA algorithm uses the genetic algorithm for local structure editing, hence is expected to work well on optimizing properties that are

Optimizing QED			
Method	Similarity	QED-Improve	Success
JTVAE	0.30±0.09	0.17±0.12	17.4%
VJTNN	0.37±0.11	**0.20±0.05**	37.6%
GCPN	0.32±0.14	**0.20±0.09**	26.5%
SELFIES-GA+D	0.43±0.17	0.17±0.11	42.5%
MIMOSA	**0.50±0.30**	**0.20±0.14**	**47.8%**
Optimizing DRD			
Method	Similarity	DRD-Improve	Success
JTVAE	0.31±0.07	0.34±0.17	25.6%
VJTNN	0.36±0.09	0.40±0.20	40.5%
GCPN	0.30±0.07	0.35±0.20	27.8%
SELFIES-GA+D	0.46±0.14	0.25±0.10	37.5%
MIMOSA	**0.57±0.29**	**0.43±0.29**	**48.3%**
Optimizing PLogP			
Method	Similarity	PLogP-Improve	Success
JTVAE	0.30±0.09	0.28±0.17	2.9%
VJTNN	0.38± 0.08	0.47±0.24	14.3%
GCPN	0.32± 0.07	0.33±0.19	7.8%
SELFIES-GA+D	0.53±0.15	**0.99±0.54**	92.8%
MIMOSA	**0.56± 0.17**	0.94±0.47	**94.0%**

Table 5.3: Single-property optimization.



(a) Input Molecule X; QED:0.72; PLogP:-3.60

(b) sim: 0.66; QED: 0.93; PLogP: -1.2

(c) sim: 0.59; QED: 0.93; PLogP: -1.1

(d) sim: 0.57; QED: 0.92; PLogP:-1.6

(e) Input Molecule X; QED: 0.71; PLogP: -3.9

(f) sim: 0.837; QED: 0.90; PLogP: -0.6

(g) sim: 0.872; QED: 0.89; PLogP: -1.2

(h) sim: 0.812; QED: 0.88; PLogP: -1.4

Figure 5.2: Examples of "QED & PLogP" optimization. (Upper), the imidazole ring in the input molecule (a) is replaced by less polar rings thiazole (b and c) and thiadiazol (d). Since more polar indicates lower PLogP, the output molecules increase PLogP while maintaining the molecular scaffold. (Lower), the PLogP of input molecule (e) is increased by neutralizing the ionized amine (g) or replacing it with substructures with less electronegativity (f and h). These changes improve the QED.

sensitive to local structural changes, such as joint optimizing both QED and PLogP where PLogP is related to the polarity of a molecule and is sensitive to the change of local structure. Because of the local editing of GA, GA does not perform well on optimizing both DRD and PLogP since DRD is less sensitive to the change of local structures.

Since most baseline models were designed to optimize single drug properties, we also conduct experiments to compare MIMOSA with them on optimizing the following single properties: (1) DRD; (2) QED and (3) PLogP.

From the results shown in Table Table 5.3, we can see that when optimizing a single drug property, MIMOSA still achieved the best performance overall, with 12.5% relative higher success rate in optimizing QED compared with the second best model GA, and 28.8% relative higher success rate in optimizing both DRD compared with the second best algorithm VJTNN. Among the baseline models, algorithms such as JTVAE, VJTNN, and GCPN that were designed to optimize single property have good performance in property improvement as expected. However, they generate molecules that have lower similarity hence the final success rates. Also, GA has the lowest QED and DRD improvement may be due to its limitation in capturing global properties. High similarity between the output and input molecules is a unique requirement for the molecule optimization task, on which MIMOSA significantly outperformed the other baselines.

Case Study. To further examine how MIMOSA can also effectively improve properties that are sensitive to local structural change, e.g., PLogP, we show two examples in Figure Figure 5.2. For the first row, the imidazole ring in the input molecule (a) is replaced by less polar five-member rings thiazole (b and c) and thiadiazol (d). Since PLogP is related to the polarity of a molecule: more polar indicates lower PLogP. The generation results in the increase of PLogP while maintaining the molecular scaffold. For the second row, the PLogP of input molecule (e) is increased by neutralizing the ionized amine (g) or replacing it with substructures with less electronegativity (f and h). These changes would also help improve

the drug-likeness, i.e., QED value.

Performance impact of pre-training Note that the mGNN reaches $97\%+$ ROC-AUC in multiclass classification and bGNN achieves more than 0.99 in terms of ROC-AUC. We also observe that the quality of generated molecules is insensitive to the classification performance of mGNN and bGNN. The goal of mGNN and bGNN pretrained on large molecule datasets is to provide good representation to guide the MCMC generation process. Given the large sampling space, the risk of generating the input molecule is practically non-existing, especially since you have multiple samples to choose from.

5.5 Conclusion and Discussion

In this work, we proposed MIMOSA, a new MCMC sampling-based method for molecule optimization. MIMOSA pretrains GNNs and employs three basic substructure operations to generate new molecules and associated weights that can encode multiple drug property constraints, upon which we accept promising molecules for the next iteration. MIMOSA iteratively produces new molecule candidates and can efficiently draw molecules that satisfy all constraints. MIMOSA significantly outperformed several state-of-the-art baselines for molecule optimization with 28.5% to 49.1% improvement when optimizing PLogP+QED and PLogP+DRD, respectively.

CHAPTER 6

SIPF: SAMPLING METHOD FOR INVERSE PROTEIN FOLDING

6.1 Research Challenge

The desired 3D structure with a useful function was first observed in many protein engineering applications before identifying the amino acid sequence. The task becomes designing an amino acid sequence that can properly fold into the desired 3D structure [58, 32]. The problem is named *inverse protein folding*, which often requires machine learning models. To predict the target amino acid sequence sequentially, different network architectures have been proposed to represent 3D protein structures, e.g., multiple structured transformers with multi-head self-attention components [32], three-dimensional convolutional neural network (3DCNN) [33, 34], graph convolutional network (GCN)[35]. However, several challenges remain in these existing methods.

(C1) Lack of uncertainty quantification in the protein space: Generative models are mainly based on maximum likelihood learning, which uses point estimation and has difficulty quantifying the uncertainty.

(C2) Incremental generation degradation during sequential generation: auto-regressive generative models generate amino acid sequences sequentially. The later amino acids depend heavily on the previous ones. Thus the error may accumulate during generation.

6.2 Main Idea and Contribution

The main contributions of the proposed method are

1. **Uncertainty quantification** (address C1): To quantify uncertainty, we formulate inverse protein folding as a Markov Chain Monte Carlo (MCMC) sampling problem, where pretrained neural networks are used as MCMC proposal distribution

(Sec subsection 6.3.1).

2. **Adaptive sampling** (address C2): we design an adaptive sampling method to sample more thoroughly at the variables[1] with high uncertainty. The designed sampler allows random and weighted scans over all the amino acids and provides more flexibility than the sequential generation (Sec subsection 6.3.2).

3. **Experiment**. We conduct thorough experiments to show the superiority of SIPF, which obtains 7.4% relative improvement in the recovery rate and 6.4% relative reduction in perplexity (Sec section 6.4).

6.3 SIPF Framework

In this section, we describe sampling method for inverse protein folding (SIPF).

First, we formulate the inverse protein folding task. We represent a protein with three structures. We use $\mathbf{S} = (\mathbf{s}_1, \cdots, \mathbf{s}_N)$ to denote the amino acid sequence. The length is N, $\mathbf{s}_i$ represents the i-th token (i.e., amino acid) in the sequence. We use $\mathbf{Z} = (\mathbf{z}_1, \cdots, \mathbf{z}_N)$ to denote the secondary structure sequence. The length is also N, and $\mathbf{z}_i$ represents the kind of secondary structure to which i-th amino acid belongs. We use $\mathcal{G} = (g_1, \cdots, g_N)$ to denote the 3D graph structure of the protein. There are N nodes in the graph, g_i represents the 3D coordinate of the i-th node. Given 3D graph structure $\mathcal{G}$, inverse protein folding is to find an amino acid sequence $\mathbf{S} = (\mathbf{s}_1, \cdots, \mathbf{s}_N)$ ($\mathbf{s}_i \in V$) that maximize the likelihood function $P(\mathbf{S}|\mathcal{G})$,

$$\underset{\mathbf{S}=(\mathbf{s}_1,\cdots,\mathbf{s}_N)}{\arg\max} \quad P(\mathbf{S}|\mathcal{G}), \tag{6.1}$$

6.3.1 MCMC Proposal distribution: Pretrained Neural Networks

This section describes MCMC proposal distribution $Q_\theta(\mathbf{s}_i|\mathbf{S}_{-i}, \mathcal{G})$, which is a mixture of two pretrained neural networks, including (1) equivariant graph neural network (geometric

[1] In this paper, a variable corresponds to an amino acid.

Table 6.1: Mathematical notations and descriptions.

Notations	Descriptions
N	number of amino acids in proteins.
V	set of all the amino acids.
g_i	coordinate of the i-th node in 3D graph $\mathcal{G}$.
$\mathcal{G} = (g_1, \cdots, g_N)$	3D graph structure (N nodes with coordinates).
$\mathbf{s}_i \in V$	The i-th amino acid in the sequence $\mathbf{S}$.
$\mathbf{S} = (\mathbf{s}_1, \cdots, \mathbf{s}_N)$	Sequence of amino acid (length N).
$\mathbf{S}_{<i}$	the first $i-1$ amino acids in the sequence $\mathbf{S}$.
$\mathbf{S}_{-i} = (\mathbf{s}_1, \cdots, \mathbf{s}_{i-1}, \mathbf{s}_{i+1}, \cdots, \mathbf{s}_N)$	amino acid sequence without i-th amino acid
$P(\mathbf{S}\|\mathcal{G})$	target distribution of $\mathbf{S}$ given $\mathcal{G}$.
$Q_\theta(\mathbf{s}_i\|\mathbf{S}_{-i}, \mathcal{G})$	MCMC proposal distribution, conditional prob. of $\mathbf{s}_i$
$\|z\|$	l_2 norm of the vector z
L_1	Number of layers in EGNN.
$\mathbf{m}_i^{(l)}$	message vector of node i at l-th layer;
$\mathbf{m}_{ij}^{(l)}$	message vector of edge from i to j at l-th layer;
$\mathbf{e}_i^{(l)}$	i-th node's embedding at l-th layer;
$\mathbf{x}_i^{(l)}$	i-th node's position embeddings at l-th layer;
$H_1(\cdot), H_2(\cdot), H_3(\cdot)$	embedding in BERT.
L_2	Number of transformer layers in BERT.
γ	hyperparameter
$\mathbf{1}(\cdot)$	Indicator function
Distance$(\cdot, \cdot)$	Distance between two amino acid sequences
$\mathbf{q} = [q_1, \cdots, q_N]$	sampling weight for each amino acid, $\sum_{i=1}^N q_i = 1$.
$\lambda > 0$	hyperparameter in optimizing q

graph level); (2) BERT model (amino acid sequence level). Both models are pretrained in a self-supervised manner [57]: predicting the category of masked node/token (i.e., amino acid) conditioned on the remaining variables (nodes/tokens/amino acids) and 3D graph structure.

Equivariant Graph Neural Network (EGNN) for 3D geometric graph We leverage the state-of-the-art equivariant graph neural network (EGNN) proposed in [59]. It is translation-, rotation- and reflection-invariant with respect to an input set of 3D points. Node embeddings at the l-th layer are $\{\mathbf{e}_i^{(l)}\}_{i=1}^N$, where $l = 0, 1, \cdots, L_1$, L_1 is number of layers in EGNN. The

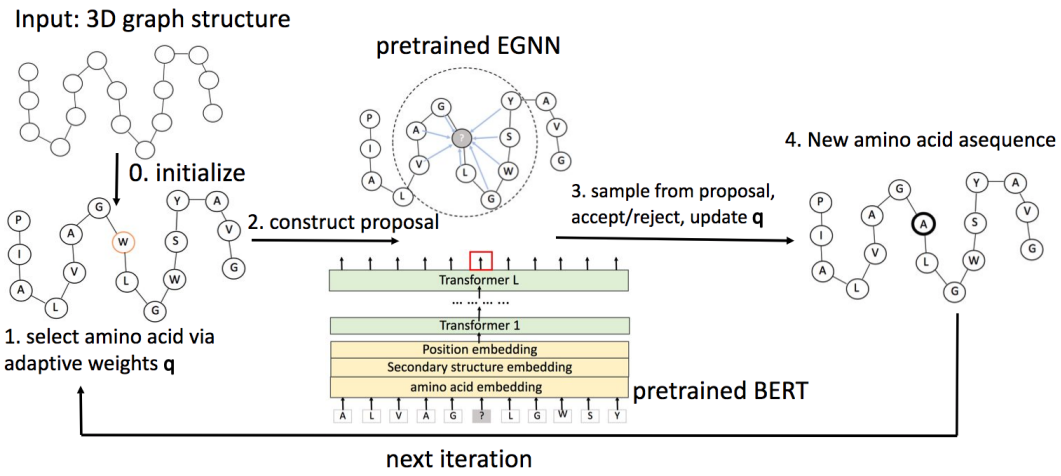


Figure 6.1: SIPF pipeline.

initial node embeddings $\{\mathbf{e}_i^{(0)}\}_{i=1}^N$ embed the categories of amino acids and are randomly initialized. Coordinate embeddings at the l-th layer are denoted $\{\mathbf{x}_i^{(l)}\}_{i=1}^N$. The initial coordinate embeddings $\{\mathbf{x}_i^{(0)}\}_{i=1}^N$ are the real 3D coordinates of all the nodes, i.e., $\{g_i\}_{i=1}^N$. The following equation defines the update rule at the l-th layer ($l = 1, \cdots, L_1$):

$$\begin{aligned}
\mathbf{m}_{ij}^{(l+1)} &= \text{MLP}_1\big(\mathbf{e}_i^{(l)}, \mathbf{e}_j^{(l)}, ||\mathbf{x}_i^{(l)} - \mathbf{x}_j^{(l)}||^2\big), \\
\mathbf{e}_i^{(l+1)} &= \text{MLP}_3\big(\mathbf{e}_i^{(l)}, \mathbf{m}_i^{(l+1)}\big), \\
\mathbf{x}_i^{(l+1)} &= \mathbf{x}_i^{(l)} + \sum_{j \neq i} \big(\mathbf{x}_i^{(l)} - \mathbf{x}_j^{(l)}\big)\text{MLP}_2\big(\mathbf{m}_{ij}^{(l)}\big), \\
\mathbf{m}_i^{(l+1)} &= \sum_j \mathbf{m}_{ij}^{(l+1)},
\end{aligned} \tag{6.2}$$

where $\text{MLP}_1(\cdot), \text{MLP}_2(\cdot), \text{MLP}_3(\cdot)$ are all two-layer multiple layer perceptrons (MLPs). Within the l-th layer, $\mathbf{m}_{ij}^{(l)}$ represent the message vector for the edge from node i to node j; $\mathbf{m}_i^{(l)}$ represents the message vector for node i, $\mathbf{x}_i^{(l)}$ is the position embedding for node i; $\mathbf{e}_i^{(l)}$ is the node embedding for node i. When the target node is $\mathbf{s}_i$, we attach an MLP structure to the last layer's node embedding of $\mathbf{s}_i$ to build EGNN proposal distribution,

$$\text{EGNN}(\mathbf{s}_i|\mathbf{S}_{-i}, \mathcal{G}) = \text{MLP}_4(\mathbf{e}_i^{(L_1)}), \tag{6.3}$$

where $\text{MLP}_4(\cdot)$ is two-layer MLP with softmax activation in the output layer. In summary, EGNN can leverage local geometric structures.

BERT model for amino acid sequence. BERT can model long range dependency in the amino acid and secondary structure sequence. Specifically, for $l = 0, \cdots, L_2$, we have

$$\begin{aligned} &\mathbf{v}_1^{(l+1)}, \cdots, \mathbf{v}_N^{(l+1)} = \text{Transformer}\big(\mathbf{v}_1^{(l)}, \cdots, \mathbf{v}_N^{(l)}\big), \\ &\mathbf{v}_i^{(0)} = \big[H_1(\mathbf{s}_i) \oplus H_2(\mathbf{z}_i) \oplus H_3(i)\big], \end{aligned} \tag{6.4}$$

where $\oplus$ denotes the concatenation of vectors; $H_1(\mathbf{s}_i)$ is the amino acid level embedding to represent $\mathbf{s}_i$; $H_2(\mathbf{z}_i)$ is the secondary structure level embedding; $H_3(i)$ is the position embedding to represent the position i [60]. The proposal distribution of i-th node is $\mathbf{v}_i^{(L_2)}$,

$$\text{BERT}(\mathbf{s}_i|\mathbf{S}_{-i}, \mathcal{G}) = \text{MLP}(\mathbf{v}_i^{(L_2)}), \tag{6.5}$$

EGNN and BERT focus on local geometric structure and long-range dependency, respectively. To combine two proposal distributions (Equation (Equation 6.3) and (Equation 6.5)), we use linear interpolation to get a better proposal distribution as follows,

$$Q_\theta(\mathbf{s}_i|\mathbf{S}_{-i}, \mathcal{G}) = \gamma\, \text{EGNN}(\mathbf{s}_i|\mathbf{S}_{-i}, \mathcal{G}) + (1-\gamma)\, \text{BERT}(\mathbf{s}_i|\mathbf{S}_{-i}, \mathcal{G}), \tag{6.6}$$

where $0 < \gamma < 1$ is a hyperparameter that controls the weights of two proposal distributions.

6.3.2 Adaptive Sampling

Then we design an adaptive sampling scheme that selects $\mathbf{s}_1, \cdots, \mathbf{s}_N$ (amino acids) with adaptive weights. We want to draw independent samples and enlarge the distance between two consecutive samples. The learning objective becomes

$$\underset{\mathbf{q}=[q_1,\cdots,q_N]}{\arg\max} \sum_{i=1}^{N} q_i D(i) + \lambda \frac{1}{N} \ln q_i, \text{ such that } \sum_{i=1}^{N} q_i = 1, \tag{6.7}$$

where $\lambda > 0$ is a hyperparameter, where $D(i)$ is the probability of position i being changed. The first term aims to maximize the expected distance between consecutive samples; while the second term serves as a regularizer that encourages the adaptive weight's distribution to be close to uniform distribution.

6.4 Main Results

Dataset and Preprocessing. **RCSB**. We download all the protein data in pdb format from https://www.rcsb.org/. The Protein Data Bank (PDB) file format is a textual file format describing the three-dimensional structures of molecules held in the Protein Data Bank. The PDB format accordingly provides description and annotation of protein and nucleic acid structures, including atomic coordinates, secondary structure assignments, and atomic connectivity. The list of all the amino acids, secondary structure, and their frequencies are provided in the supplementary materials. We collect 27,043 proteins with a single chain, from which we randomly select 1,000 proteins as a test set. The remaining proteins are used for learning. We split training and validation sets with a 9:1 ratio. The training and validation set contains 24,338 and 2,705 proteins, respectively.

CATH [61] is a dataset based on the hierarchical classification of protein structure (CATH) available at https://www.cathdb.info/, also in PDB format. Following [32, 36, 25], for all domains in the CATH 4.2 40% non-redundant set of proteins, we collect full chains up to length 500 and then randomly assign their CATH topology classifications (CAT codes) to train, validation and test sets at a targeted 8/1/1 split. Since each chain can contain multiple CAT codes, we first removed any redundant entries from train and then from validation. Finally, we removed any chains from the test set that had CAT overlap with train and removed chains from the validation set with CAT overlap to train or test. This resulted in a dataset of 15,802 chains in the training set, 1,975 chains in the validation set, and 1,887 chains in the test set.

Baseline. For all baselines, we use the default setup (hyperparameter) in the original papers. (i) **StructTrans (Structured Transformer)** [32] uses three layers of self-attention and position-wise feedforward modules for the encoder and decoder; (ii) **ProDCoNN (Protein design convolutional neural network)** [33] uses a gridded box centered on the target residue to capture the local structural information. The atoms and their features are later voxelized into the 3D voxel grid. A 3D convolutional layer followed by a max-pooling layer is then attached, followed by an MLP layer to make a prediction; (iii) **DeepGCN (Deep Graph Convolutional Network)** [35] used graph convolutional network to represent the node and edge attributes, where the 3D graph is transformed into a 2D graph with an adjacency matrix. (iv) **Fold2Seq (Protein Folding to Sequence)** [36] jointly learns a sequence embedding using a transformer and a fold embedding from the density of secondary structural elements in 3D voxels. Traditional physics-based method RosettaDesign [62] performs much worse than state-of-the-art deep learning methods and is inefficient [32, 33, 36, 63]. Thus, it is not included in the baselines. For reference, we also show the results of (i) **Uniform (Uniform frequencies)**: random amino acid sequence under the uniform distribution of all the amino acids and (ii) **Natural (Natural frequencies)**: random amino acid sequence through natural frequencies of amino acids. We calculate the natural frequencies of all the amino acids on the processed protein data and report them in the supplementary materials.

Evaluation Metrics. We use the following metrics to evaluate the performance of all the methods, following [32, 33, 34, 35]. We use the evaluation metrics following [32, 33, 34, 35]. **(1) Recovery rate (RR)** (%): percentage of correctly recovered amino acids in the whole sequence; **(2) Perplexity (PPL)** measures how well a probability model can predict a protein. Lower perplexities indicate better performance. **(3) Amino acid level accuracy (AAA)**. For each amino acid, the prediction can be seen as a binary classification task (correctly recovered or not), we report average Precision-Recall Area Under the Curve

(PR-AUC) over all the amino acids as metrics to measure the accuracy on the amino acid level.

Table 6.2: Experimental results on RCSB and CATH. The results are averages and standard deviations of 5 independent runs. On each metric, we highlight the best score and use * to denote the results pass the t-test (SIPF versus Fold2Seq, the best baseline) with p-value $<$ 0.05. The t-test results show that improvements of SIPF over the best baseline method are significant in most of the metrics on both tasks.

	Method	RR (↑)	PPL (↓)	AAA (↑)
RCSB	Uniform	5.52±0.13%	20.02±0.06	0.15±0.01
	Natural	9.18±0.08%	17.44±0.07	0.21±0.01
	StructTrans	29.81±0.15%	9.30±0.11	0.40±0.02
	ProDCoNN	25.78±0.25%	9.92±0.21	0.35±0.02
	DeepGCN	28.00±0.24%	9.67±0.11	0.38±0.01
	Fold2Seq	30.20±0.24%	9.28±0.10	0.43±0.01
	SIPF	**32.43±0.23%***	**8.69±0.13***	**0.46±0.01***
	Method	RR (↑)	PPL (↓)	AAA (↑)
CATH	Uniform	5.13±0.04%	20.03±0.05	0.15±0.01
	Natural	9.84±0.05%	17.50±0.04	0.20±0.01
	StructTrans	28.56±0.08%	9.47±0.06	0.38±0.01
	ProDCoNN	26.52±0.11%	9.85±0.10	0.36±0.01
	DeepGCN	27.78±0.12%	9.71±0.11	0.38±0.01
	Fold2Seq	30.02±0.11%	9.38±0.05	0.43±0.01
	SIPF	**31.45±0.13%***	**8.72±0.10***	**0.45±0.01**

The performance of all the compared methods on RCSB and CATH are presented in Table Table 6.2. We observe that our method achieves the highest recovery rate (RR), amino acid-level accuracy (AAA), and lowest (best) perplexity among all the compared methods on both datasets. Specifically, compared with the best baseline method Fold2Seq, our method achieves 7.4% relative improvement on recovery rate (RR) (30.20% v.s. 32.43%) and 6.4% relative reduction in perplexity (8.69 v.s. 9.28) on RCSB, 4.5% relative improvement on recovery rate (RR) (30.02% v.s. 31.45%) and 7.0% relative reduction in perplexity (8.72 v.s. 9.38) on CATH. The results of hypothesis testing (t-test) show the improvements over the best baseline method are significant in most of the metrics.

Table 6.3: Ablation studies.

Method	RR (↑)	PPL (↓)	AAA (↑)
EGNN only	32.17±0.23%	8.71±0.13	0.45±0.01
BERT only	29.80±0.27%	9.47±0.11	0.40±0.01
Gibbs sampling	28.81±0.14%	9.62±0.12	0.39±0.02
uniform sampling	31.50±0.15%	9.28±0.12	0.42±0.02
w.o. reject	31.75±0.21%	9.20±0.16	0.43±0.02
SIPF	**32.43±0.23%**	**8.69±0.13**	**0.46±0.01**

Ablation Study. To further understand our method, we conduct an ablation study on the RCSB dataset to investigate the impact of each component on the performance. Specifically, we explore the empirical effect for both the conditional probability-based proposal and sampling method and consider the following variants of our method.

(1) EGNN only. Our MCMC proposal distribution is a mixture of EGNN and BERT prediction, as described in Equation (Equation 6.6). The variant uses only EGNN prediction as a proposal to consider the local geometric structural information only, i.e., $\gamma = 1$ (Equation Equation 6.6 in Section subsection 6.3.1). **(2) BERT only**. The variant uses only BERT prediction as MCMC proposal distribution to consider the long-range dependency only, i.e., $\gamma = 0$ (Equation Equation 6.6 in Section subsection 6.3.1). **(3) Gibbs sampling**. Gibbs sampling scans all the variables in a fixed order [64], instead of adaptive sampling in our method (Section subsection 6.3.2). **(4) uniform sampling**. The variant randomly and uniformly selects all the variables instead of adaptive weight. **(5) w.o. reject**. The variant does not reject the proposal but accepts all the proposals. It studies the effect of leveraging approximate target distribution. To make the comparison fair, the total numbers of sampling iterations for all the methods are the same, ten times of the amino acid sequence length.

Table Table 6.3 reports the results of the ablation study, which demonstrates the best performance of the full method SIPF. From the first two lines, we find that both EGNN and BERT have positive contributions to the performance. We observe that removing EGNN causes the most degradation in both recovery rate and perplexity, suggesting that EGNN is more important than BERT. This is also validated by the fact that $\gamma = 0.7$ achieves

the best performance by putting more weight on the EGNN component. Comparing the 3rd, 4th, and last lines, we also observe that adaptive sampling in SIPF outperforms Gibbs sampling and uniform sampling. In addition, comparing the last two lines, we find if we accept the proposal, the performance will degrade, demonstrating the positive contribution of approximate target distribution $\widetilde{P}(\mathbf{S})$. In sum, the MCMC proposal, adaptive sampling, and approximate target distribution are all key components of SIPF.

6.5 Conclusion and Discussion

In this paper, to address the challenges in the existing inverse protein folding methods, we have proposed a sampling-based method for inverse protein folding. Concretely, we first formulate it as a sampling problem and then design two pretrained neural networks as (conditional probability) MCMC proposal distribution. We also design a novel sampling method (an adaptive sampling scheme and approximate target distribution) to quantify uncertainty and enhance exploration ability to data space. Thorough empirical studies are conducted to confirm the superiority of the proposed method SIPF.

Part III

Differentiable Programming

Overview In the third part of the thesis, we discuss differentiable programming. Specifically, the drug molecules are usually regarded as discrete structured data. Modeling discrete object relies on combinatorial optimization more or less and suffers from a brute-force trial-and-error strategy, which is computationally prohibitive. To address this issue, we propose *differentiable programming* that throws the discrete optimization problem into a continuous optimization problem. The main idea of differentiable programming is to relax the discrete object into a continuous domain, which enables the gradient-based optimization to manipulate the molecule object directly and circumvent the brute-force trial-and-error strategy.

Differentiable programming can be used in small-molecule drug design, where we design a differentiable scaffolding tree (a high-level abstract of the molecular graph) to convert the molecular optimization from a combinatorial optimization problem into a continuous optimization problem that can be solved efficiently via gradient descent.

> chapter 7: **Differentiable Scaffolding Tree for Molecular Optimization.** Tianfan Fu*, Wenhao Gao*, Cao Xiao, Jacob Yasonik, Connor W. Coley, Jimeng Sun. International Conference on Learning Representation (ICLR), 2022.

Therapeutic antibodies have become one of the fastest-growing classes of drugs and have been approved for the treatment of a wide range of indications, from cancer to autoimmune diseases. Complementarity-determining regions (CDRs) are part of the variable chains in antibodies and determine specific antibody-antigen binding. The key to the antibody design is to design the CDR loop because the remaining part of the antibody is relatively fixed. Thus, we also focus on using deep generative models to design an antibody CDR loop. Unlike small-molecule, the CDR loop is a 3D structure with some constraints on the 3D structure. Based on the idea of differentiable programming, we design a Constrained Energy Model for Antibody Complementarity Determining Regions (CDRs) design to encode these constraints.

chapter 8: **Antibody Complementarity Determining Regions (CDRs) design using Constrained Energy Model.** Tianfan Fu, Jimeng Sun. The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (2022).

CHAPTER 7

DIFFERENTIABLE SCAFFOLDING TREE FOR MOLECULAR OPTIMIZATION

7.1 Research Challenge

Recent advances in deep generative models (DGM) for molecule optimization allow learning the distribution of molecules and optimizing the latent embedding vectors of molecules. Models in this category are exemplified by the variational autoencoder (VAE) [10, 11]. On the other hand, because of the discrete and not explicitly combinatorial nature of the enormous chemical space, applying combinatorial optimization algorithms with some structure enumeration has been the predominant approach [26, 30, 29]. Deep learning models have also been used to guide these combinatorial optimization algorithms. For example, [26, 28] tried to solve the problem with deep reinforcement learning; [15] approached the problem via MCMC sampling guided by graph neural networks. Despite the initial success of these previous attempts, the following challenges remain:

- **C1**: deep generative models optimize the molecular structures in a learned latent space, which requires the latent space to be smooth and discriminative. Training such models needs carefully designed networks and well-distributed datasets.

- **C2**: most combinatorial optimization algorithms, featured by evolutionary learning methods [29, 30, 65, 15], exhibit random-walk behavior, and leverage trial-and-error strategies to explore the discrete chemical space. The recent deep reinforcement learning methods [26, 27, 28] aim to remove random-walk search using a deep neural network to guide the searching. It is challenging to design the effective reward function into the objective [28].

- **C3**: Most existing methods require a great number of oracle calls (a property evaluator)

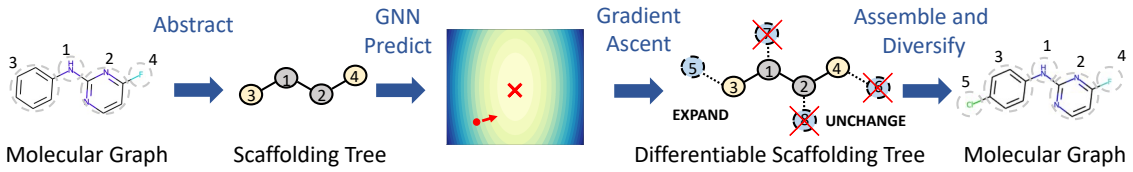


Figure 7.1: Illustration of Differentiable Scaffolding Tree approach. Convert molecular graph to differentiable scaffolding tree. We show non-leaf nodes (grey), leaf nodes (yellow), and expansion nodes (blue). The dashed nodes and edges are learnable.

to proceed with an efficient search. However, realistic oracle functions, evaluating with either experiments or high-fidelity computational simulations, are usually expensive.

7.2 Main Idea and Contribution

To address these challenges, we proposed a differentiable scaffolding tree (DST) for molecular structure and the main contributions are:

- We propose the differentiable scaffolding tree to define a local derivative of a chemical graph. This concept enables a gradient-based optimization of a discrete graph structure. It is the first attempt to make the molecular optimization problem differentiable at the structure level rather than resorting to latent spaces or using RL/evolutionary algorithms.

- We present a general molecular optimization strategy utilizing local derivatives defined by differentiable scaffolding tree. This strategy leverages the property landscape's geometric structure and suppresses random-walk behavior, exploring chemical space more efficiently. We also incorporate a determinantal point process (DPP) based selection strategy to enhance the diversity of generated molecules.

- DST demonstrates encouraging preliminary results on *de novo* molecular optimization and requires fewer oracle calls.

7.3 DST Framework

In this section, we describe Differentiable Scaffolding Tree (DST). The mathematical notations are listed in Table Table 7.1 for ease of exposition.

We first introduce the formulation of molecular optimization and *differentiable scaffolding tree* (DST) in Section subsection 7.3.1, illustrate the pipeline in Figure Figure 7.1, then describe the key steps following the order:

- **Oracle GNN**: We use Oracle GNN to replace black box oracle. Oracle GNN is trained once and for all. The training is separate from optimizing DST below.
- **Optimizing differentiable scaffolding tree:** We formulate the discrete molecule optimization into *a locally differentiable* problem with a differentiable scaffolding tree (DST). Then a DST can be optimized by the gradient back-propagated from oracle GNN.
- **Molecule Diversification** After that, we describe how we design a *determinantal point process (DPP)* based method to output diverse molecules for iterative learning.

7.3.1 Problem Formulation and Notations

Oracle $\mathcal{O}$ is a black-box function that evaluates certain chemical or biological properties. Suppose we want to optimize P molecular properties specified by oracle $\mathcal{O}_1, \cdots, \mathcal{O}_P$, we formulate molecule optimization problem as

$$\arg\max_{X \in \mathcal{Q}} \; F(X; \mathcal{O}_1, \mathcal{O}_2, \cdots, \mathcal{O}_P) = f(\mathcal{O}_1(X), \cdots, \mathcal{O}_P(X)), \tag{7.1}$$

where X is a molecule, $\mathcal{Q}$ denotes the set of valid molecules; f is the composite objective combining all the oracle scores, e.g., the mean value of P oracle scores. A scaffolding tree, $\mathcal{T}_X$, is a spanning tree whose nodes are substructures. $\mathcal{T}_X$ is represented by (i) node indicator matrix, (ii) adjacency matrix, and (iii) node weight vector. Among the K nodes in $\mathcal{T}_X$, there

are K_{leaf} leaf nodes and $K - K_{\text{leaf}}$ non-leaf nodes. The sets of leaf nodes and non-leaf nodes are denoted $\mathcal{V}_{\text{leaf}}$ and $\mathcal{V}_{\text{nonleaf}}$ correspondingly.

Table 7.1: Mathematical Notations.

Notations	Descriptions
$\mathcal{O}$	Oracle function, e.g., evaluator of molecular property.
F	objective function of molecule generation (Equation Equation 7.1).
$P \in \mathbb{N}_+$	Number of target oracles.
$\mathcal{Q}$	Set of all the valid chemical molecules.
$\mathcal{S}$	Vocabulary set, i.e., substructure set. A substructure is an atom or a ring.
$\mathcal{T}$	Scaffolding tree.
$K = \lvert\mathcal{T}\rvert$	number of nodes in scaffolding tree $\mathcal{T}$.
$\mathbf{N}; \mathbf{A}; \mathbf{w}$	Node indicator matrix; adjacency matrix; node weight.
$\mathcal{V}_{\text{leaf}}$	Leaf node set in scaffolding tree $\mathcal{T}$.
$\mathcal{V}_{\text{nonleaf}}$	Nonleaf node set in scaffolding tree $\mathcal{T}$.
$\mathcal{V}_{\text{expand}}$	Expansion node set in scaffolding tree $\mathcal{T}$.
$K_{\text{leaf}} = \lvert\mathcal{V}_{\text{leaf}}\rvert$	Size of leaf node set.
$K_{\text{expand}} = \lvert\mathcal{V}_{\text{expand}}\rvert = K$	Size of expansion node set. $K_{\text{leaf}} = K_{\text{expand}}$.
$d \in \mathbb{N}_+$	GNN hidden dimension.
$L \in \mathbb{N}_+$	GNN depth.
$\Theta = \{\mathbf{E}\} \cup \{\mathbf{B}^{(l)}, \mathbf{U}^{(l)}\}_{l=1}^{L}$	Learnable parameter of GNN.
$\mathbf{E} \in \mathbb{R}^{\lvert\mathcal{S}\rvert \times d}$	embedding stackings of all the substructures in vocabulary set $\mathcal{S}$.
$\mathbf{B}^{(l)} \in \mathbb{R}^{K \times d}$	bias parameters at l-th layer.
$\mathbf{U}^{(l)} \in \mathbb{R}^{d \times d}$	weight parameters at l-th layer.
$\mathbf{H}^{(l)}, l = 0, \cdots, L$	Node embedding at l-th layer of GNN
$\mathbf{H}^{(0)} = \mathbf{NE} \in \mathbb{R}^{K \times d}$	initial node embeddings, stacks basic embeddings of all the nodes in the scaffolding tree.
MLP	multilayer perceptron
ReLU	ReLU activate function
$\hat{y}$	GNN prediction.
y	groundtruth
$\mathcal{L}$	Loss function of GNN.
$\mathcal{D}$	the training set
$\mathcal{N}(X)$	Neighborhood molecule set of X (Def 3).
Λ	differentiable edge set.
$\widetilde{\mathbf{N}}; \widetilde{\mathbf{A}}; \widetilde{\mathbf{w}}$	Differentiable node indicator matrix; adjacency matrix; node weight.
$\det()$	Determinant of a square matrix
$M \in \mathbb{N}_+$	Number of all possible molecules to select.
$C \in \mathbb{N}_+$	Number of selected molecules.
$\mathbf{S} \in \mathbb{R}_+^{M \times M}$	Similarity kernel matrix.
$\mathbf{V} \in \mathbb{R}_+^{M \times M}$	Diagonal scoring matrix.
$\mathcal{R}$	subset of $\{1, 2, \cdots, M\}$, index of select molecules.
$\lambda > 0$	hyperparameter that balances desirable property and diversity.

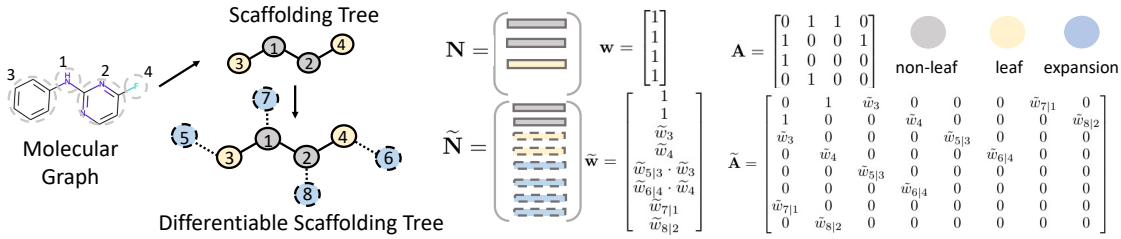


Figure 7.2: Example of differentiable scaffolding tree.

Differentiable scaffolding tree

Similar to a scaffolding tree, a differentiable scaffolding tree (DST) also contains (i) node indicator matrix, (ii) adjacency matrix, and (iii) node weight vector, but with additional expansion nodes. To make it locally differentiable, we modify the tree parameters from two aspects: (A) *node identity* and (B) *node existence*. First, We enable optimization on the node indicator:

Definition 1. ***Differentiable node indicator matrix*** $\widetilde{\mathbf{N}}$ *takes the form:*

$$\widetilde{\mathbf{N}} = \begin{pmatrix} \widetilde{\mathbf{N}}_{nonleaf} \\ \widetilde{\mathbf{N}}_{leaf} \\ \widetilde{\mathbf{N}}_{expand} \end{pmatrix} \in \mathbb{R}_{+}^{(K+K_{expand})\times|\mathcal{S}|}, \quad \sum_{j=1}^{|\mathcal{S}|} \widetilde{\mathbf{N}}_{ij} = 1, \quad K = K_{expand}. \tag{7.2}$$

$\widetilde{\mathbf{N}}_{nonleaf} = \mathbf{N}_{nonleaf} \in \{0,1\}^{(K-K_{leaf})\times|\mathcal{S}|}$ *are fixed, equal to the part in the original scaffolding tree, each row is a one-hot vector, indicating that we fix all the non-leaf nodes. In contrast, both* $\widetilde{\mathbf{N}}_{expand}$ *and* $\widetilde{\mathbf{N}}_{leaf}$ *are learnable, we use softmax activation to implicitly encode the constraint* $\sum_j \widetilde{\mathbf{N}}_{ij} = 1$ *i.e.,* $\widetilde{\mathbf{N}}_{ij} = \frac{\exp(\widehat{\mathbf{N}_{ij}})}{\sum_{j'=1}^{|\mathcal{S}|}\exp(\widehat{\mathbf{N}_{i,j'}})}$*,* $\widehat{\mathbf{N}}$ *are the parameters to learn. This constraint guarantees that each row of* $\widetilde{\mathbf{N}}$ *is a valid substructures' distribution.*

Also, We enable optimization on *node existence* by assigning learnable weights for the leaf and expansion nodes, constructing an adjacency matrix and node weight vector:

Definition 2. ***Differentiable adjacency matrix*** $\widetilde{\mathbf{A}} \in \mathbb{R}^{(K+K_{expand})\times(K+K_{expand})}$ *takes the form:*

$$\widetilde{\mathbf{A}}_{ij} = \widetilde{\mathbf{A}}_{ji} = \begin{cases} \sigma(\widehat{\mathbf{w}}_i), & (i,j) \in \Lambda, i \in \mathcal{V}_{leaf},\ j \in \mathcal{V}_{nonleaf} \\ \sigma(\widehat{\mathbf{w}}_{i|j}), & (i,j) \in \Lambda, i \in \mathcal{V}_{expand},\ j \in \mathcal{V}_{leaf} \cup \mathcal{V}_{nonleaf}, \end{cases} \tag{7.3}$$

where Λ *is the differentiable edge set defined above, Sigmoid function* $\sigma(\cdot)$ *imposes the constraint* $0 \leq \widetilde{\mathbf{A}}_{ij} \leq 1$. $\widehat{\mathbf{w}} \in \mathbb{R}^{K_{leaf}+K_{expand}}$ *are the parameters.*

Then we construct a differentiable surrogate model to capture the knowledge from any oracle function. We choose graph neural network architecture for its state-of-the-art performance in modeling structure-property relationships. In particular, we imitate the objective function F with GNN (graph convolutional network [9]):

$$\hat{y} = \text{GNN}(X;\Theta) \approx F(X;\mathcal{O}_1,\mathcal{O}_2,\cdots,\mathcal{O}_P) = y, \tag{7.4}$$

where Θ represents the GNN's parameters. Concretely, we use a graph convolutional network (GCN) [9]. The initial node embeddings $\mathbf{H}^{(0)} = \mathbf{NE} \in \mathbb{R}^{K\times d}$ stacks basic embeddings of all the nodes in the scaffolding tree, d is the GCN hidden dimension, $\mathbf{N}$ is the node indicator matrix. $\mathbf{E} \in \mathbb{R}^{|\mathcal{S}|\times d}$ is the embedding matrix of all the substructures in vocabulary set $\mathcal{S}$, and is randomly initialized. The updating rule of GCN for the l-th layer is

$$\mathbf{H}^{(l)} = \text{RELU}\big(\mathbf{B}^{(l)} + \mathbf{A}(\mathbf{H}^{(l-1)}\mathbf{U}^{(l)})\big), \quad l = 1,\cdots,L, \tag{7.5}$$

where L is GCN's depth, $\mathbf{A}$ is the adjacency matrix, $\mathbf{H}^{(l)} \in \mathbb{R}^{K\times d}$ is the nodes' embedding of layer l, $\mathbf{B}^{(l)}$ and $\mathbf{U}^{(l)} \in \mathbb{R}^{d\times d}$ are bias and weight parameters, respectively. Then we optimize Differentiable Scaffolding Tree via local editing.

Local Editing Operations For a leaf node v in the scaffolding tree, we can perform three editing operations,

1. **SHRINK**: delete node v;

2. **REPLACE**: replace a new substructure over v;

3. **EXPAND**: add a new node u_v that connects to node v.

For a nonleaf node v, we support

1. **EXPAND**: add a new node u_v connecting to v;

2. **do nothing**.

If we EXPAND and REPLACE, the new substructures are sampled from the vocabulary $\mathcal{S}$. We define the molecule neighborhood set:

Definition 3 (Neighborhood set)**.** *Neighborhood set of molecule X, denoted $\mathcal{N}(X)$, is the set of all the possible molecules obtained by imposing one local editing operation to scaffolding tree $\mathcal{T}_X$ and assembling the edited trees into molecules.*

Optimizing DST. Then within the domain of neighborhood molecule set $\mathcal{N}(X)$, the objective function can be represented as a differentiable function of X's DST ($\widetilde{\mathbf{N}}_X, \widetilde{\mathbf{A}}_X, \widetilde{\mathbf{w}}_X$). We address the following optimization problem to get the best scaffolding tree within $\mathcal{N}(X)$,

$$\widetilde{\mathbf{N}}_*, \widetilde{\mathbf{A}}_*, \widetilde{\mathbf{w}}_* = \arg\max_{\{\widetilde{\mathbf{N}}_X, \widetilde{\mathbf{A}}_X, \widetilde{\mathbf{w}}_X\}} \ \text{GNN}(\{\widetilde{\mathbf{N}}_X, \widetilde{\mathbf{A}}_X, \widetilde{\mathbf{w}}_X\}; \Theta_*), \tag{7.6}$$

where the GNN parameters Θ_* are fixed. It is differentiable with regard to $\{\widetilde{\mathbf{N}}, \widetilde{\mathbf{A}}, \widetilde{\mathbf{w}}\}$ for all molecules in the neighborhood set $\mathcal{N}(X)$. DST pipeline leverages iterative local discrete search. In t-th iteration, we optimize the DST of $X^{(t)}$, i.e., X in Equation (Equation 7.6) is $X^{(t)}$.

Sampling from DST. Then we sample the new scaffolding tree from the optimized DST. Concretely, for each leaf node $v \in \mathcal{V}_{\text{leaf}}$ and the corresponding expansion node $u_v \in \mathcal{V}_{\text{expand}}$,

we select one of the following steps with probabilities (w.p.) as follows,

$$\mathcal{T} \sim \text{ DST-Sampler}(\widetilde{\mathbf{N}}_*, \widetilde{\mathbf{A}}_*, \widetilde{\mathbf{w}}_*)$$
$$= \begin{cases} \text{1. SHRINK: delete leaf node } v, & \text{w.p. } 1-(\widetilde{\mathbf{w}}_*)_v, \\ \text{2. EXPAND: add } u_v\text{, select substructure at } u_v \text{ based on } (\widetilde{\mathbf{N}}_*)_{u_v}, & \text{w.p. } (\widetilde{\mathbf{w}}_*)_v(\widetilde{\mathbf{w}}_*)_{u_v|v}, \\ \text{3. REPLACE: select substructure at } v \text{ based on } (\widetilde{\mathbf{N}}_*)_u, & \text{w.p. } (\widetilde{\mathbf{w}}_*)_v(1-(\widetilde{\mathbf{w}}_*)_{u_v|v}). \end{cases} \tag{7.7}$$

Assemble. Each scaffolding tree corresponds to multiple molecules due to the multiple ways substructures can be combined. Once the scaffolding tree is produced, we can efficiently enumerate all the possible molecules following [11].

7.3.2 Molecule Diversification

In the current iteration, we have generated M molecules $(X_1, \cdots, X_M)$ and need to select C molecules for the next iteration. We expect these molecules to have desirable chemical properties (high F score) and simultaneously maintain higher structural diversity. To do so, we resort to the *determinantal point process (DPP)* [66], which models the repulsive correlation between data points. Specifically, for M data points, whose indexes are $\{1, 2, \cdots, M\}$, $\mathbf{S} \in \mathbb{R}_+^{M\times M}$ denotes the similarity matrix between these data points. To create a diverse subset (denoted $\mathcal{R}$) with fixed size C, the sampling probability should be proportional to the determinant of the submatrix $\mathbf{S}_\mathcal{R} \in \mathbb{R}^{C\times C}$, i.e., $P(\mathcal{R}) \propto \det(\boldsymbol{S}_\mathcal{R})$, where $\mathcal{R} \subseteq \{1, 2, \cdots, M\},\ |\mathcal{R}| = C$. Combining the objective (F) value and diversity, the composite objective is

$$\underset{\mathcal{R}\subseteq\{1,2,\cdots,M\},|\mathcal{R}|=C}{\arg\max} \mathcal{L}_{\text{DPP}}(\mathcal{R}) = \lambda \sum_{r\in\mathcal{R}} F(X_r) + \log P(\mathcal{R}) = \log\det(\boldsymbol{V}_\mathcal{R}) + \log\det(\boldsymbol{S}_\mathcal{R}), \tag{7.8}$$

where the hyperparameter $\lambda > 0$ balances the two terms, the diagonal scoring matrix $\boldsymbol{V} = \text{diag}([\exp(\lambda F(X_1)), \cdots, \exp(\lambda F(X_M))])$, $\boldsymbol{V}_\mathcal{R} \in \mathbb{R}^{C\times C}$ is a sub-matrix of $\mathbf{V}$ indexed by $\mathcal{R}$. When λ goes to infinity, it is equivalent to selecting top-C candidates with the highest

F score, same as conventional evolutionary learning in [30, 29]. Inspired by generalized DPP methods [67], we further transform $\mathcal{L}_{\text{DPP}}(\mathcal{R})$,

$$\mathcal{L}_{\text{DPP}}(\mathcal{R}) = \log\det(\boldsymbol{V}_{\mathcal{R}}) + \log\det(\boldsymbol{S}_{\mathcal{R}}) = \log\det\left(\boldsymbol{V}_{\mathcal{R}}^{\frac{1}{2}}\boldsymbol{S}_{\mathcal{R}}\boldsymbol{V}_{\mathcal{R}}^{\frac{1}{2}}\right) = \log\det\left((\boldsymbol{V}^{\frac{1}{2}}\mathbf{S}\boldsymbol{V}^{\frac{1}{2}})_{\mathcal{R}}\right).$$

where $\boldsymbol{V}^{\frac{1}{2}}\mathbf{S}\boldsymbol{V}^{\frac{1}{2}}$ is symmetric positive semi-definite. Then it can be solved by generalized DPP methods in $O(C^2M)$ [67]. The computational complexity of DST is $O(TMC^2)$.

Then, molecule diversification can be transformed as

$$\underset{\mathcal{R}\subseteq\{1,2,\cdots,M\},|\mathcal{R}|=C}{\arg\max}\ \mathcal{L}_{\text{DPP}}(\mathcal{R}) = \log\det\left(\left(\boldsymbol{V}^{\frac{1}{2}}\mathbf{S}\boldsymbol{V}^{\frac{1}{2}}\right)_{\mathcal{R}}\right), \tag{7.9}$$

7.4 Main Results

First, we briefly describe the basic experimental setup, mainly following [11, 18, 26, 21, 28, 65]. **Task**. We focus on the following two optimization tasks: (1) **molecular modification** (2) **de novo molecule generation**.

Following [28, 65] the target molecular properties, **Molecular Properties** contains **QED**; **LogP**; **SA**; **JNK3**; **GSK3β**, following [28, 29, 68, 65], where QED quantifies drug-likeness; LogP indicates the water-octanol partition coefficient; SA stands for synthetic accessibility and is used to prevents the formation of chemically unfeasible molecules; JNK3/GSK3β measure inhibition against c-Jun N-terminal kinase-3/Glycogen synthase kinase 3 beta. For all 5 scores (including normalized SA), higher is better. We conducted (1) single-objective generation that optimizes JNK3, GSK3β and LogP separately and (2) multi-objective generation that optimizes the mean value of "JNK3+GSK3β" and "QED+SA+JNK3+GSK3β" in the main text.

Dataset: ZINC 250K contains around 250K druglike molecules [45, 69]. To select the substructure set $\mathcal{S}$, we break the molecules into substructures (including single rings and single atoms), We select the substructures that appear more than 1000 times in ZINC 250K

as the vocabulary set $\mathcal{S}$, which contains 82 most frequent substructures.

Baselines. (1) **LigGPT** (string-based distribution learning model with Transformer as a decoder) [70]; (2) **GCPN** (Graph Convolutional Policy Network) [26]; (3) **MolDQN** (Molecule Deep Q-Network) [27]; (4) **GA+D** (Genetic Algorithm with Discriminator network) [29]; (5) **MARS** (Markov Molecular Sampling) [65]; (6) **RationaleRL** [28]; (7) **ChemBO** (Chemical Bayesian Optimization) [71]; (8) **BOSS** (Bayesian Optimization over String Space) [68]. Among them, LigGPT belongs to the deep generative model, where all the oracle calls can be precomputed; GCPN and MolDQN are deep reinforcement learning methods; GA+D and MARS are evolutionary learning methods; RationaleRL is a deep generative model fine-tuned with RL techniques. ChemBO and BOSS are Bayesian optimization methods. We also consider a DST variant: **DST-rand**. Instead of optimizing and sampling from DST, DST-rand leverages random local search, i.e., randomly selecting basic operations (EXPAND, REPLACE, SHRINK) and substructure from the vocabulary. To improve efficiency, we also select a subset of all the random samples with high surrogate GNN prediction scores. All the baselines except LigGPT require online oracle calls.

Metrics. We consider the following metrics (1) **Novelty (Nov)** (% of molecules not in training set); (2) **Diversity (Div)** (average pairwise Tanimoto distance); (3) **Average Property Score (APS)** (average top-100 molecules); (4) **# of oracle calls**: DST needs to call oracle in labeling data for GNN (**precomputed**) and DST based *de novo* generation (**online**), we show the costs for both steps. For each method in Table Table 7.2 and Table 7.3, we set the number of oracle calls so that the property score nearly converges w.r.t. oracle call's number. Since we only enumerate valid chemical structures during the recovery from scaffolding trees, the chemical validities are always 100%.

Optimization Performance The de novo generation is to design novel, diverse molecules with desirable chemical properties from scratch. We consider (1) single-objective generation that optimizes JNK3, GSK3β and LogP separately and (2) multi-objective generation that

Table 7.2: Multi-objective *de novo* design. #oracle = (1)"**precomputed** oracle call" (to label molecules in existing database) + (2)"**online** oracle call" (during learning).

Method	JNK3+GSK3β				QED+SA+JNK3+GSK3β			
	Nov↑	Div↑	APS↑	#oracle↓	Nov↑	Div↑	APS↑	#oracle↓
LigGPT	**100%**	**0.845**	0.271	100k+0	**100%**	**0.902**	0.378	100k+0
GCPN	**100%**	0.578	0.293	0+200K	**100%**	0.596	0.450	0+200K
MolDQN	**100%**	0.605	0.348	0+200K	**100%**	0.597	0.365	0+200K
GA+D	**100%**	0.657	0.608	0+50K	97%	0.681	0.632	0+50K
RationaleRL	**100%**	0.700	0.795	25K+67K	99%	0.720	0.675	25K+67K
MARS	**100%**	0.711	0.789	0+50K	**100%**	0.714	0.662	0+50K
ChemBO	98%	0.702	0.747	0+50K	99%	0.701	0.648	0+50K
BOSS	99%	0.564	0.504	0+50K	98%	0.561	0.504	0+50K
DST-rand	**100%**	0.456	0.622	10+5K	**100%**	0.765	0.575	20K+5K
DST	**100%**	0.750	**0.827**	**10K+5K**	**100%**	0.755	**0.752**	**20K+5K**

optimizes the mean value of "JNK3+GSK3β" and "QED+SA+JNK3+GSK3β".

The results are shown in Table Table 7.3. We find that deep generative model (LigGPT) and RL-based methods (GCPN and MolDQN) fail in some tasks, which is consistent with the results reported in MARS [65]. Overall, DST obtains the best results in most tasks. In terms of success rate and diversity, DST outperformed all baselines in most tasks. These results show our gradient-based optimization strategy has a strong optimization ability to provide a diverse set of molecules with high objective functions.

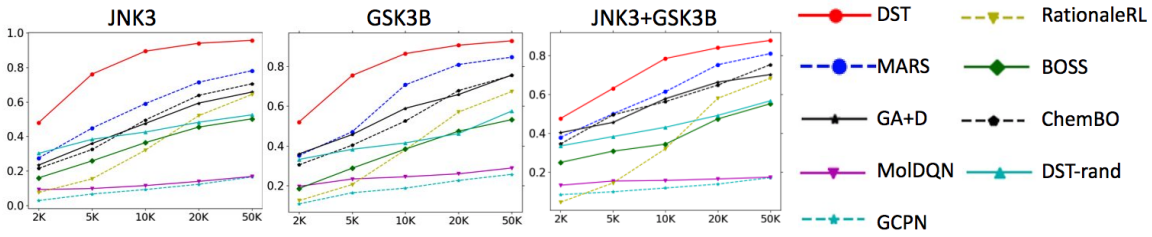


Figure 7.3: Oracle efficiency test. Top-100 average score v.s. the number of oracle calls.

We can see that the majority of de novo optimization methods require oracle calls online (instead of pre-computation), including all RL/evolutionary algorithm-based baselines. DST takes fewer oracle calls compared with baselines. DST can leverage the precomputed oracle calls to label the molecules in an existing database (i.e., ZINC) for training the oracle GNN

Table 7.3: Single-objective *de novo* molecular generation.

Method	JNK3				GSK3β				LogP			
	Nov↑	Div↑	APS↑	#oracle↓	Nov↑	Div↑	APS↑	#oracle↓	Nov↑	Div↑	APS↑	#oracle↓
LigGPT	**100%**	**0.837**	0.302	100K+0	**100%**	**0.867**	0.283	100K+0	**100%**	**0.868**	4.56	100K+0
GCPN	**100%**	0.584	0.365	0+200K	**100%**	0.519	0.400	0+200K	**100%**	0.532	5.43	0+200K
MolDQN	**100%**	0.605	0.459	0+200K	**100%**	0.545	0.398	0+200K	**100%**	0.485	6.00	0+200K
GA+D	99%	0.702	0.615	0+50K	98%	0.687	0.678	0+50K	**100%**	0.721	30.2	0+50K
RationaleRL	99%	0.681	0.803	25K+32K	99%	0.731	0.806	30K+45K	-	-	-	-
MARS	**100%**	0.711	0.784	0+50K	**100%**	0.735	0.810	0+50K	**100%**	0.692	44.1	0+30K
ChemBO	98%	0.645	0.648	0+50K	98%	0.679	0.492	0+50K	98%	0.732	10.2	0+50K
DST-rand	**100%**	0.754	0.413	10K+10K	97%	0.793	0.455	10K+10K	**100%**	0.713	36.1	10K+15K
DST	**100%**	0.732	**0.928**	**10K+5K**	**100%**	0.748	**0.869**	**10K+5K**	**100%**	0.704	**47.1**	**10K+5K**

and dramatically saving the oracle calls during reference. In the three tasks in Table Table 7.3, two-thirds of the oracle calls (10K) can be precomputed or collected from other sources. To further verify the oracle efficiency, we explore a special setting of molecule optimization where the budget of oracle calls is limited to a fixed number (2K, 5K, 10K, 20K, 50K) and compare the optimization performance. For GCPN, MolDQN, GA+D, and MARS, the learning iteration number depends on the budget of oracle calls. RationaleRL [28] is not included because it requires intensive oracle calls to collect enough reference data, exceeding the oracle budget in this scenario. In DST, we use around 80% budget to label the dataset (i.e., training GNN) while the remaining budget is to conduct *de novo* design. Specifically, for 2K, 5K, 10K, 20K, and 50K, we use 1.5K, 4K, 8K, 16K, and 40K oracle calls to label the data for learning GNN, respectively. We show the average objective values of top-100 molecules under different oracle budgets in Figure Figure 7.3. Our method shows a significant advantage compared to all the baseline methods in all limited budget settings. It is also worth mentioning that our method can even perform oracle-free optimization, as we can use the trained GNN as a static surrogate, though sacrificing some optimization ability. In real cases, labeled data are usually collected for expensive oracles. We can train the GNN with those data and optimize the molecules without explicitly calling the oracle.

7.5 Conclusion and Discussion

This paper proposed Differentiable Scaffolding Tree (DST) to make a molecular graph locally differentiable, allowing a continuous gradient-based optimization. To the best of our

knowledge, it is the first attempt to make the molecular optimization problem differentiable at the substructure level rather than resorting to latent spaces or using RL/evolutionary algorithms. We constructed a general molecular optimization strategy based on DST, corroborated by thorough empirical studies.

CHAPTER 8
ANTIBODY COMPLEMENTARITY DETERMINING REGIONS (CDRS) DESIGN USING CONSTRAINED ENERGY MODEL

8.1 Research Challenge

Most of the affinity and specificity of antibodies are modulated by a set of binding loops called the Complementarity Determining Regions (CDRs) found on the variable domain of antibodies. There is a high demand to develop *in silico* methods for antibody design, especially CDR loop design [72]. Recently machine learning methods have been proposed in designing novel antibodies [73, 42]. However, several challenges remain:

C1. Antibody CDR loops have specific geometry shape [74]. However, most of the existing antibody design methods do not consider it, which may generate invalid CDR loops.

C2. Most of the existing deep generative models do not leverage external knowledge and are purely learning from data, impeding their ability to incorporate constraints.

8.2 Main Idea and Contribution

To address these issues, we proposed Constrained Energy Model that considers geometry constraints during the generation of 3D CDR loops. **The main contributions** are:

(1) We formulate antibody CDR design as a constrained 3D generation task and define a *constrained manifold* to represent all the geometric valid CDR loops.

(2) We design a *Constrained Energy Model* that learns the 3D structure on the defined manifold (Section subsection 8.3.2).

(3) Experimental results confirm the effectiveness of the proposed method, which obtains up to 33.4% relative reduction in 3D geometry error (Root Mean Square Deviation, RMSD) and 8.4% relative improvement in terms of amino acid sequence metric (perplexity)

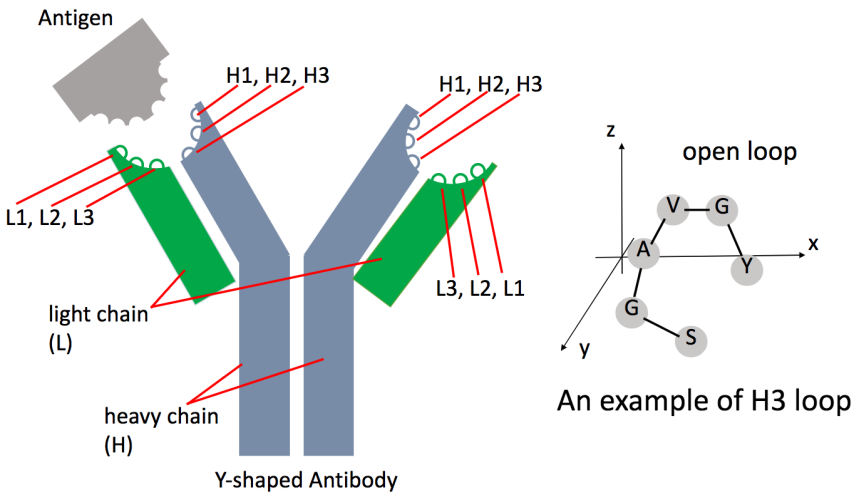


Figure 8.1: Antibody structure.

(Section section 8.4). We provide more details as follows.

8.3 Constrained Energy Model Framework

In this section, we describe Constrained Energy Model (CEM).

8.3.1 CDR Loop Design

We first introduce the *de novo* antibody CDR loop generation problem. Amino acids are the basic building blocks of proteins. The set of amino acids is denoted $\mathcal{V}$, which contains 20 natural amino acids. Proteins consist of one or more chains of amino acids called *polypeptides* [72]. The sequences of the amino acid chain cause the polypeptide and are folded into a three-dimensional (3D) functional shape. An antibody is a special kind of protein that is symmetric and Y-shaped. In the Y-shaped antibody, there are six CDR loops, L1, L2, L3 loops on the light chain and H1, H2, H3 loops on the heavy chain [75, 76]. A 3D CDR loop (H1, H2, or H3) can be characterized by a sequence of amino acids and their 3D coordinates. A CDR loop is denoted $\mathcal{Y}$, suppose it has N amino acids, it is represented as

$$\mathcal{Y} = (\mathcal{A}, \mathcal{X}), \quad \mathcal{A} = [\mathbf{a}_1, \cdots, \mathbf{a}_N], \ \mathcal{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_N]. \tag{8.1}$$

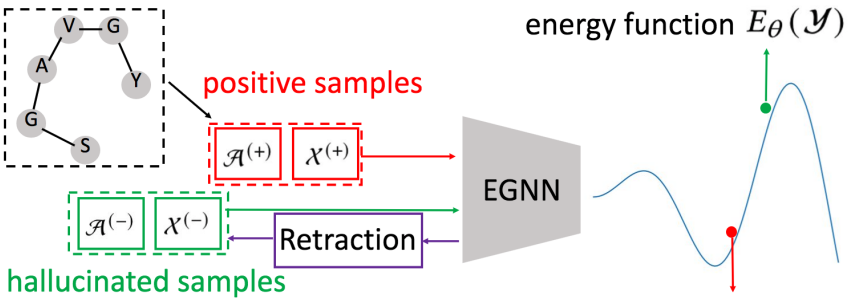


Figure 8.2: Pipeline of CEM.

Compared with L1, L2, and L3 loops, the H1, H2, and H3 loop in the CDR of an antibody plays a critical role in its binding ability to potential antigens [77, 78].

Validity constraints. We define the validity of the generated loop based on empirical domain knowledge about CDR loops [74, 79]. Specifically, we define the validity of a generated 3D CDR loop when it satisfies the following two constraints:

1. **Peptide bond length**. Multiple amino acids are linked together by peptide bonds and form a single chain. The distance between connected amino acids is a constant [74]: $||\mathbf{x}_i - \mathbf{x}_{i+1}||_2 = \kappa, \text{ for } i = 1, \cdots, N-1.$

2. **Open loop**. The shape of CDR is an open loop, as shown in Figure Figure 8.1, where the distance between the first and the last amino acids is within a specific range [79], $\epsilon_1 \leq ||\mathbf{x}_1 - \mathbf{x}_N||_2 \leq \epsilon_2$.

The setup of $\kappa, \epsilon_1, \epsilon_2$ is based on domain knowledge [79] and empirical validation. For H1 loop, $\kappa = 3.80$, $\epsilon_1 = 11.4, \epsilon_2 = 13.1$; for H2 loop, $\kappa = 3.81$, $\epsilon_1 = 5.0, \epsilon_2 = 5.9$; for H3 loop, $\kappa = 3.81$, $\epsilon_1 = 6.50, \epsilon_2 = 8.50$. We define manifold $\mathcal{M}$ to represent the constraints.

$$\mathcal{M} = \left\{(\mathcal{A}, \mathcal{X}) \mid \ ||\mathbf{x}_i - \mathbf{x}_{i+1}||_2 = \kappa, \epsilon_1 \leq ||\mathbf{x}_1 - \mathbf{x}_N||_2 \leq \epsilon_2\right\}. \tag{8.2}$$

Constrained *de novo* antibody CDR loop design aims to generate novel 3D CDR loops $\mathcal{Y}$s within the constrained manifold $\mathcal{M}$ from scratch, i.e., $\mathcal{Y} \in \mathcal{M}$.

8.3.2 Constrained Energy Model (CEM)

The Constrained Energy Model defines a parameterized probability distribution P_θ over all the CDR loops $\mathcal{Y}$ in the constrained manifold $\mathcal{M}$, $\mathcal{M}$ is constrained manifold that contains all the geometric valid CDR loops,

$$P_\theta(\mathcal{Y}) = \frac{e^{-E_\theta(\mathcal{Y})}}{Z(\theta)}, \quad \mathcal{Y} \in \mathcal{M}, \quad Z(\theta) = \int_{\mathcal{Y}\in\mathcal{M}} e^{-E_\theta(\mathcal{Y})} d\mathcal{Y} \tag{8.3}$$

the data ($\mathcal{Y}$) with lower energy E_θ corresponds to higher probability/likelihood P_θ. the normalization constant $Z(\theta)$ is computationally intractable. To represent the 3D CDR loop, we leverage the state-of-the-art equivariant graph neural network (EGNN) proposed in [59].

The core idea of training the energy model is to push down the positive samples and push up the hallucinated samples at the same time. In the Constrained Energy Model, we restrict hallucinated samples in the manifold $\mathcal{M}$. The learning objective is

$$\arg\max_\theta \mathbb{E}_{\mathcal{Y}^{(+)}\sim P_{\text{data}}}\big[-E_\theta(\mathcal{Y}^{(+)})\big] + \mathbb{E}_{\mathcal{Y}^{(-)}\sim P_\theta}\big[E_\theta(\mathcal{Y}^{(-)})\big], \quad \mathcal{Y}^{(+)}, \mathcal{Y}^{(-)} \in \mathcal{M}, \tag{8.4}$$

Specifically, gradient methods do not need to evaluate $P_\theta(\mathcal{Y})$ directly. Instead, it needs to evaluate the gradient of the log probability, i.e., $-\nabla E_\theta$. At the t-th step, it is updated via

$$\begin{aligned} \big(\mathcal{A}^{(-)}\big)^{(t)} &= \big(\mathcal{A}^{(-)}\big)^{(t-1)} - \lambda_t \nabla_{\mathcal{A}} E_\theta((\mathcal{A}^{(-)})^{(t-1)}) + \boldsymbol{\Xi}, \\ \big(\mathcal{X}^{(-)}\big)^{(t)} &= \mathcal{R}_{\mathcal{M}}\big(\big(\mathcal{X}^{(-)}\big)^{(t-1)} - \lambda_t \nabla_{\mathcal{X}} E_\theta((\mathcal{X}^{(-)})^{(t-1)}) + \boldsymbol{\Gamma}\big), \end{aligned} \tag{8.5}$$

where λ_t is the step size at the t-th iteration, $\boldsymbol{\Xi}$ and $\boldsymbol{\Gamma}$ have the same shape with $\mathcal{A}$ and $\mathcal{X}$, respectively. Each scalar element in $\boldsymbol{\Xi}/\boldsymbol{\Gamma}$ is i.i.d. drawn from zero-mean Gaussian distribution whose variance is $\sqrt{\lambda_t}$, i.e., $\boldsymbol{\Gamma_i} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\lambda_t})$ for any scalar element $\Gamma_i \in \boldsymbol{\Gamma}$, $\boldsymbol{\Xi_i} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\lambda_t})$ for any scalar element $\boldsymbol{\Xi_i} \in \boldsymbol{\Xi}$. Our sampling space is restricted to manifold $\mathcal{M}$, when update $\mathcal{X}$, we project the updated samples to the manifold using *retraction* operation (denoted $\mathcal{R}_{\mathcal{M}}$), following [80].

8.4 Main Results

We conducted generation tasks on H1, H2, and H3 loops and reported the results in Table Table 8.1. We conducted 5 independent runs with different random seeds, reported average results, and have these observations:

1. **our method outperforms all the baseline methods significantly** in terms of amino acid sequence level metric (PPL, perplexity) and geometry graph metrics (RMSD, root-mean-square deviation, and %V, validity rate). Specifically, compared with the best baseline method (IR-GNN), on H1/H2/H3 design tasks, our method achieves 14.9%/15.0%/26.2% relative improvement in terms of %V respectively, 24.1%/11.6%/33.4% relative reduction in terms of RMSD respectively, and 7.6%/3.5%/8.4% relative reduction in terms of PPL;

2. **H1 v.s. H2 v.s. H3**: among all the three kinds of design tasks, including H1, H2, and H3 loops, almost all the methods get the highest perplexity, RMSD, and lower validity in the H3 generation task. This is consistent with the existing knowledge that CDR H3 loops have the highest variability and are most challenging to design [42].

3. **Diversity**: EBM, CEM, IR-GNN, and AR-GNN perform similarly in terms of diversity, validating that our method can explore the amino acid sequence space thoroughly. The diversity is measured on the amino acid sequence level.

Ablation study (effect of constraints). To show the empirical effect of constraints, we also compare the results of EBM in Table Table 8.1. We observe that CEM outperforms the vanilla energy-based model significantly and consistently across all three generation tasks (H1, H2, and H3), obtaining 32.8%, 120.7%, 57.9% relative improvement in terms of validity rate (% V), respectively. The key reason behind this observation is: CEM constrains the learning space to the constrained manifold $\mathcal{M}$, and only needs to discriminate (i.e., assigning lower or higher energy value) the data points on the manifold. The constrained

Table 8.1: *de novo* antibody CDR loop (including H1, H2, H3) design results on SAbDab.

Task	Method	PPL (↓)	RMSD (↓)	%V (↑)	Div (↑)
H1	Reference	8.10±0.08	0.0±0.00	100.0±0.0%	0.518±0.024
	LSTM	10.20±0.23	-	-	0.553±0.045
	GA	10.48±0.26	1.99±0.13	44.0±1.9%	0.635±0.047
	AR-GNN	9.55±0.25	1.97±0.11	61.7±1.7%	0.632±0.050
	IR-GNN	9.18±0.16	1.70±0.11	87.0±1.3%	0.684±0.014
	EBM	9.84±0.27	1.92±0.25	75.3±1.8%	**0.691±0.028**
	CEM	**8.48±0.19***	**1.29±0.15***	**100.0±0.0%***	0.684±0.010
H2	Reference	8.57±0.12	0.0±0.0	100.0±0.0%	0.603±0.015
	LSTM	10.86±0.35	-	-	0.633±0.030
	GA	10.25±0.26	1.93±0.19	34.3±1.8%	0.544±0.050
	AR-GNN	10.54±0.24	1.69±0.30	34.5±1.1%	**0.671±0.038**
	IR-GNN	9.65±0.16	1.12±0.17	86.9±0.9%	0.618±0.023
	EBM	10.00±0.39	1.44±0.30	45.3±1.0%	0.665±0.031
	CEM	**9.31±0.10***	**0.99±0.11**	**100.0±0.0%***	0.664±0.025
H3	Reference	9.84±0.32	0.0±0.0	100.0±0.0%	0.745±0.031
	LSTM	12.35±0.33	-	-	0.736±0.047
	GA	12.75±0.29	4.33±0.98	13.4±%	0.713±0.054
	AR-GNN	13.01±0.13	3.80±0.52	25.8±0.9%	0.754±0.025
	IR-GNN	11.45±0.25	3.02±0.24	78.4±0.7%	0.751±0.017
	EBM	10.93±0.48	3.21±0.86	62.7±1.0%	**0.798±0.043**
	CEM	**10.49±0.15***	**2.01±0.10***	**99.0±0.3%***	0.786±0.013

Energy Model is more efficient than the unconstrained energy model (i.e., vanilla energy model) in terms of sample complexity.

8.5 Conclusion and Discussion

We have proposed Constrained Energy Model (CEM) for designing 3D antibody CDR loops. We first design a constrained manifold for all the CDR loops that satisfy geometry constraints. Then we design Constrained Energy Model that learns from both positive and hallucinated samples in the constrained manifold and updates hallucinated samples in the constrained manifold. Thorough empirical studies validate CEM's superiority in designing CDR H1, H2, and H3 loops.

Part IV

Intelligent Combinatorial Optimization

Overview In the last part of the thesis, we discuss intelligent combinatorial optimization methods for drug design. Specifically, the drug molecules are essentially discrete structured data objects. The most straightforward method is combinatorial optimization. However, most of the existing combinatorial optimization algorithms (e.g., genetic algorithm, Monte Carlo tree search) rely heavily on brute-force trial-and-error strategies and are always computationally expensive. To address this issue, we propose intelligent combinatorial optimization. Combinatorial optimization is essentially a search problem. The main idea of intelligent combinatorial optimization is to estimate the potential reward for each searching branch and prioritize the promising branches to search the discrete space intelligently.

Traditional combinatorial optimization methods such as genetic algorithms (GA) have demonstrated state-of-the-art performance in various drug molecular optimization tasks. However, they rely heavily on a random-walk-like exploration, which leads to unstable performance. To achieve a more stable and efficient drug design method, we propose a Reinforced Genetic Algorithm (RGA) that uses neural models to prioritize the profitable design steps and suppress random-walk behavior. We validate the effectiveness of the proposed method on small-molecule drug design.

chapter 9: **Reinforced Genetic Algorithm for Structure-based Drug Design.**
Tianfan Fu*, Wenhao Gao*, Connor W. Coley, Jimeng Sun. Neural Information Processing Systems (NeurIPS) 2022.

CHAPTER 9
REINFORCED GENETIC ALGORITHM FOR STRUCTURE-BASED DRUG DESIGN

9.1 Research Challenge

Rapid drug discovery that requires less time and cost is of significant interest in pharmaceutical science, whose importance has been highlighted in the recent pandemic. Structure-based drug design (SBDD) [81] that leverages the three-dimensional (3D) structures of the disease-related proteins to design drug candidates is one primary approach to accelerate the drug discovery processes with physical simulation and data-driven modeling. According to the lock and key model [82], the molecules that bind tighter to a disease target are more likely to expose bioactivity against the disease, which has been verified experimentally [83]. As AlphaFold2 has provided accurate predictions to most human proteins [84, 85], SBDD has a tremendous opportunity to discover new drugs for new targets that we cannot model before [86].

SBDD could be formulated as an optimization problem where the objective function is the binding affinity estimated by simulations such as docking [82]. The most widely used design method is virtual screening, which exhaustively investigates every molecule in a library and ranks them. Lyu et al. successfully discovered new chemotypes for AmpC β-lactamase and the D_4 dopamine receptor by studying hundreds of millions of molecules with docking simulation [87]. However, the number of the drug-like molecules is large as estimated to be 10^{60} [81], and it is computationally prohibitive to screen all of the possible molecules. Though machine learning approaches have been developed to accelerate screening [88, 89], it is still challenging to screen large enough chemical space within the foreseeable future.

Instead of naively screening a library, designing drug candidates with generative models has been highlighted as a promising strategy, exemplified by [90, 91]. This class of methods models the problem as the generation of ligands conditioned on the protein pockets. However, as generative models are trained to learn the distribution of known active compounds, they tend to produce molecules similar to training data [92], which discourages finding novel molecules and leads to unsatisfactory optimization performance.

A more straightforward solution is a combinatorial optimization algorithm that searches the implicitly defined discrete chemical space. As shown in multiple standard molecule optimization benchmarks [23, 25, 24], combinatorial optimization methods, especially genetic algorithms (GA) [30, 93], often perform better than deep generative models. The key to superior performance is GA's action definition. Specifically, in each generation (iteration), GA maintains a population of possible candidates (a.k.a. parents) and conducts the crossover between two candidates and mutation from a single candidate to generate new offspring. These two types of actions, crossover, and mutation, enable global and local traversal over the chemical space, allowing a thorough exploration and superior optimization performance.

However, most GA algorithms select mutation and crossover operations randomly [30], leading to significant variance between independent runs. Especially in SBDD, when the oracle functions are expensive molecular simulations, it is resource-consuming to ensure stability by running multiple times. Further, most current combinatorial methods are designed for general-purpose molecular optimization and simply use a docking simulation as an oracle. It is challenging to leverage the structure of proteins in these methods, and we need to start from scratch whenever we change a protein target, even though the physics of ligand-protein interaction is shared. Ignoring the shared information across tasks leads to unnecessary exploration steps and, thus, demands many more oracle calls, which require expensive and unnecessary simulations [94].

9.2 Main Ideas and Contributions

To overcome these issues in the GA method, we propose Reinforced Genetic Algorithm (RGA), which attempts to reformulate an evolutionary process as a Markov decision process and uses neural networks to make informed decisions and suppress the random-walk behavior. Specifically, we utilize an E(3)-equivariant neural network [59] to choose parents and mutation types based on the 3D structure of the ligands and proteins. The networks are pre-trained with various native complex structures to utilize the knowledge of the shared binding physics between different targets and then fine-tuned with a reinforcement learning algorithm during optimizations. We test RGA's performance with various disease-related targets, including the main protease of SARS-CoV-2.

The main contributions of this work can be summarized as follows:

- We propose an evolutionary Markov decision process (EMDP) that reformulates an evolutionary process as a Markov decision process, where the state is a population of molecules instead of a single molecule (Section subsection 9.3.2).
- We show the first successful attempt to use a neural model to guide the crossover and mutation operations in a genetic algorithm to suppress random-walk behavior and explore the chemical space intelligently (Section subsection 9.3.3).
- We present a structure-based de novo drug design algorithm that outperforms baseline methods consistently through thorough empirical studies on optimizing binding affinity by leveraging the underlying binding physics (Section section 9.4).

9.3 RGA framework

In this work, we focus on structure-based drug design. The goal is to design drug molecules (a.k.a. ligands) that could bind tightly with the disease-related proteins (a.k.a. targets). Given the 3D structures of the target proteins, including binding site information, docking is

a popular computational method for assessing the binding affinity, which can be roughly retrieved as the free energy changes during the binding processes. We present a variant of a genetic algorithm that is guided by reinforcement learning and a docking oracle. Next, we will first describe the general evolutionary process used in genetic algorithms (Section subsection 9.3.1); Then, we will present how to model this evolutionary process as a Markov decision process (MDP) where RL framework can be constructed (Section subsection 9.3.2); After that, we describe the detailed implementation of this MDP framework using multiple policy networks (Section subsection 9.3.3).

Docking simulation. The purpose of target-ligand docking is to find the optimal binding between a small molecule (ligand) and a target (target protein). Docking can be conducted using well-commercialized software, such as AutoDock Vina [95]. The input is a 2D molecular graph, the 3D geometric shape of target and the corresponding binding site. The output is the 3D pose and relative position of the ligand (binds to target) that corresponds to the best binding affinity score. In this work, we use X to denote the ligand (including its 3D pose that binds to the target), $\mathcal{T}$ to denote 3D target structures. The mathematical notation table is available in Appendix. In this work, we leverage one 3D pose and relative coordinates of the ligand that corresponds to the best binding affinity score.

For ease of exposition, we list the mathematical notations in Table Table 9.1. All the mathematical notations are divided into three parts: (1) notation for genetic algorithm (Section subsection 9.3.1); (2) notation for equivariance neural networks (ENN) [59] (Section subsection 9.3.3); (3) notations for policy network (Section subsection 9.3.3).

9.3.1 Evolutionary Process

In this section, we introduce the primary setting of the evolutionary processes. With both optimization performance and synthetic accessibility taken into account [97, 25], we follow the action settings in Autogrow 4.0 [93]. It demonstrated superior performance over other GA variants in the empirical validation of structure-based drug design [93], and its mutation

Table 9.1: Mathematical Notations. All the mathematical notations are divided into three parts: (1) notation for genetic algorithm (Section subsection 9.3.1); (2) notation for equivariance neural networks (ENN) [59] (Section subsection 9.3.3); (3) notations for policy network (Section subsection 9.3.3).

Notations	Descriptions
X	ligand (drug molecule, including 3D pose)
$\mathcal{T}$	target (target protein related to the disease)
$\mathcal{S}^{(t)}$	the state (population of molecule) at the t-th generation.
$\mathcal{Q}^{(t)}$	offspring pool at the t-th generation.
K	the number of molecules in the state, i.e., size of the population.
$X_{\text{crossover}}^{\text{parent 1/2}}$	the first/second parent molecule in the crossover.
$X_{\text{crossover}}^{\text{child 1/2}}$	the first/second child molecule in the crossover.
$X_{\text{mutation}}^{\text{parent}}$	parent molecule in the mutation
$X_{\text{mutation}}^{\text{child}}$	child molecule in the mutation
$\xi \in \mathcal{R}$	the selection reaction in the mutation
$\mathcal{R}$	the reaction set (library) for mutation
ENN	equivariance neural networks [59]
$\mathcal{V} = \{H, C, O, N, \cdots\}$	vocabulary set of atoms
$\mathcal{Y} = (\mathcal{A}, \mathcal{Z})$	3D structure
$\mathcal{A}$	categories of all the atoms
$\mathbf{a}_i$	one-hot vector that encode category of i-th atom
$\mathcal{Z}$	3D coordinates of the atoms
$\mathbf{D} \in \mathbb{R}^{\lvert\mathcal{V}\rvert \times d}$	the embedding matrix of all the categories of atoms
d	the hidden dimension in ENN.
N	number of atoms in the input of ENN.
L	number of layers in ENN
$l = 0, 1, \cdots, L$	index of layer in ENN
MLP	multiple layer perceptrons
$\text{MLP}_e(\cdot), \text{MLP}_x(\cdot), \text{MLP}_h(\cdot)$	two-layer MLP in ENN with Swish activation [96] in hidden layer
$\oplus$	the concatenation of vectors
$\mathbf{Z}^{(0)} = \{\mathbf{z}_i\}_{i=1}^N$	initial coordinate embeddings, real 3D coordinates of all the nodes.
$\mathbf{H}^{(l)} = \{\mathbf{h}_i^{(l)}\}_{i=1}^N$	Node embeddings at the l-th layer
$\mathbf{h}_i^{(0)} = \mathbf{D}^\top \mathbf{a}_i \in \mathbb{R}^d$	The initial node embedding that embeds the i-th node
$\mathbf{Z}^{(l)} = \{\mathbf{z}_i^{(l)}\}_{i=1}^N$	Coordinate embeddings at the l-th layer
$\mathbf{w}_{ij}^{(l)}$	message vector for the edge from node i to node j at l-th layer
$\mathbf{v}_i^{(l)}$	message vector for node i at l-th layer
$\mathbf{z}_i^{(l)}$	the position embedding for node i at l-th layer
$\mathbf{h}_i^{(l)}$	the node embedding for node i at l-th layer
$\mathbf{h}_{\mathcal{Y}} = \text{ENN}(\mathcal{Y})$	ENN representation of the 3D graph $\mathcal{Y}$ (Equation Equation 9.1)
$p_{\text{crossover}}^{(1)}(X_{\text{crossover}}^{\text{parent 1}} \lvert \mathcal{S}^{(t)})$	probability to select the first parent molecule in crossover
$p_{\text{crossover}}^{(2)}(X_{\text{crossover}}^{\text{parent 2}} \lvert X_{\text{crossover}}^{\text{parent 1}}, \mathcal{S}^{(t)})$	probability to select the second parent molecule in crossover
$p_{\text{crossover}}(X_{\text{crossover}}^{\text{child 1}}, X_{\text{crossover}}^{\text{child 2}} \lvert \mathcal{S}^{(t)})$	probability of two generated child molecules in crossover (Eq Equation 9.4)
$p_{\text{mutation}}^{(1)}(X_{\text{mutation}}^{\text{parent}} \lvert \mathcal{S}^{(t)})$	probability to select the parent molecule in mutation
$p_{\text{mutation}}^{(2)}(\xi \lvert X_{\text{mutation}}^{\text{parent}}, \mathcal{S}^{(t)})$	probability to select the reaction in mutation
$p_{\text{mutation}}(X_{\text{mutation}}^{\text{child}} \lvert \mathcal{S}^{(t)})$	probability of generated child molecule in mutation (Eq Equation 9.7)

actions originated from chemical reactions so that the designed molecules are more likely to be synthesizable. Specifically, an evolutionary process starts by randomly sampling

a *population* of drug candidates from a library. In each *generation* (iteration), it carries out (i) *crossover* between parents selected from the last generation, and (ii) *mutation* on a single child to obtain the offspring pool. Note that we only adopted the action settings from Autogrow 4.0, without using other tricks such as elitism.

Crossover, also called recombination, combines the structure of two parents to generate new children. Following Autogrow 4.0 [93], we select two parents from the last generation and search for the largest common substructure shared between them. Then we generate two children by randomly switching their decorating moieties, i.e., the side chains attached to the common substructure.

Mutation operates on a single parent molecule and modifies its structure slightly. Following Autogrow 4.0 [93], we adopt transformations based on chemical reactions. Unlike naively defined atom-editing actions, mutation steps based on chemical reactions could ensure all modification is reasonable in reality, leading to a larger probability of designing synthesizable molecules. We included two types of chemical reactions: uni-molecular reactions, which only require one reactant, and bi-molecular reactions, which require two reactants. While uni-molecular reactions could be directly applied to the parent, we sample a purchasable compound to react with the parent when conducting a bi-molecular reaction. In both cases, the parent serves as one reactant, and we use the main product as the child molecule. We use the chemical reactions from [93], which was originally from [98, 99].

Evolution. At the t-th generation (iteration), given a population of molecules denoted as $\mathcal{S}^{(t)}$, we generate an offspring pool denoted as $\mathcal{Q}^{(t)}$ by applying crossover and mutation operations. Then we filter out the ones with undesirable physical and chemical properties (e.g., poor solubility, high toxicity) in the offspring pool and select the most promising K to form the next generation pool ($\mathcal{S}^{(t+1)}$).

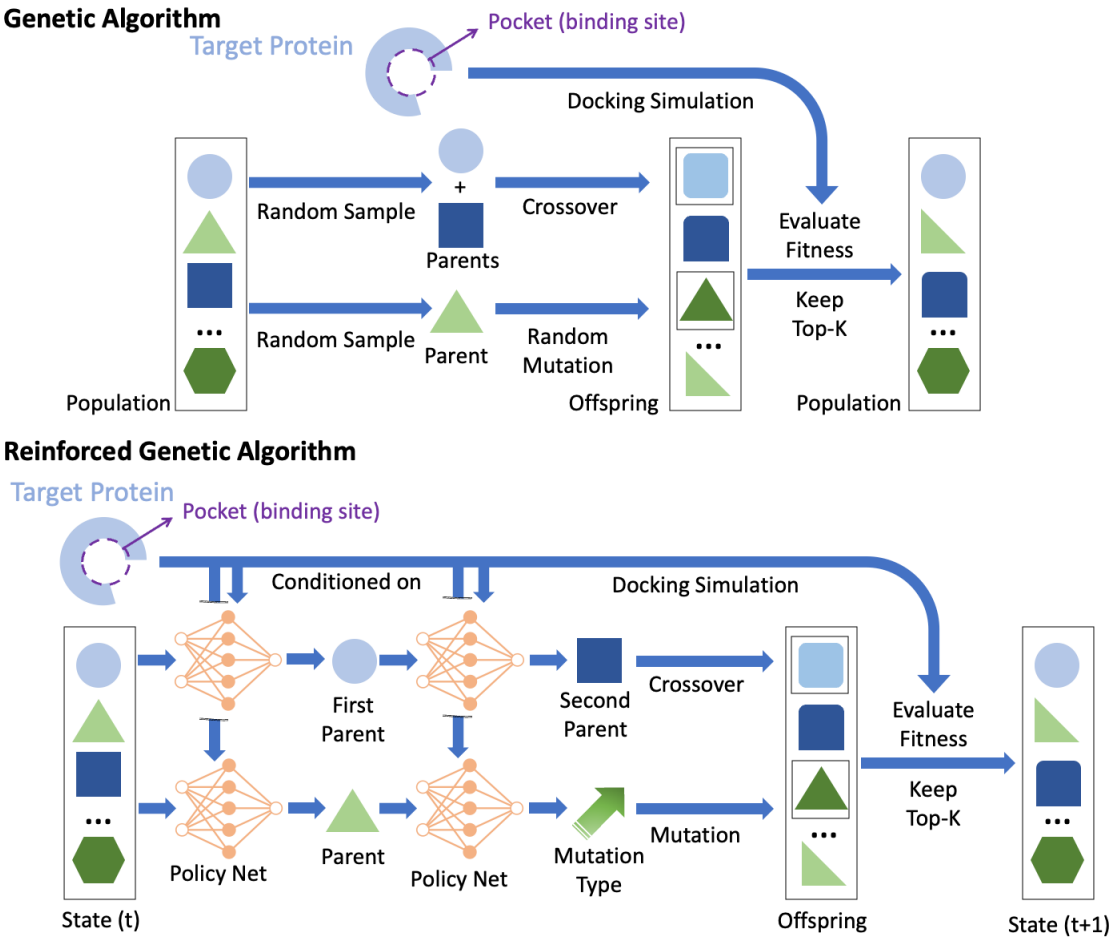


Figure 9.1: We illustrate one generation (iteration) of GA (top) and RGA pipeline (bottom). Specifically, we train policy networks that take the target and ligand as input to make informed choices on parents and mutation types in RGA.

9.3.2 Evolutionary Markov Decision Process

Next, we propose the evolutionary Markov decision process (EMDP) that formulates an evolutionary process of genetic algorithm as a Markov decision process (MDP). The primary purpose is to utilize reinforcement learning algorithms to train networks to inform the decision steps to replace random selections. Taking a generation as a state, Markov property that requires $P(\mathcal{S}^{(t+1)}|\mathcal{S}^{(1)}, \cdots, \mathcal{S}^{(t)}) = P(\mathcal{S}^{(t+1)}|\mathcal{S}^{(t)})$ is naturally satisfied by the evolutionary process described above, where $\mathcal{S}^{(t)}$ denotes the state at the t-th generation, which is the population of ligands. We use X to denote a ligand. We elaborate on essential components for the Markov decision process as follows, and the EMDP pipeline is illustrated

in Figure Figure 9.1.

State Space. We define the population at the t-step generation, $\mathcal{S}^{(t)}$, in the evolutionary process as the state at the t-step in an EMDP. A state includes a population of candidate molecules (i.e., ligand, denoted X) and their 3D poses docked to the target, fully observable to the RL agent. At the beginning of the EMDP, we randomly select a population of candidate molecules and use docking simulation to yield their 3D poses as the initial state.

Action Space. The actions in an EMDP are to conduct the two evolutionary steps: crossover and mutation, in a population. For each evolutionary step, we need two actions to conduct it. Concretely, crossover ($X^{\text{parent 1}}_{\text{crossover}}, X^{\text{parent 2}}_{\text{crossover}} \xrightarrow{\text{crossover}} X^{\text{child 1}}_{\text{crossover}}, X^{\text{child 2}}_{\text{crossover}}$) can be divided to two steps:

1. select the first candidate ligand $X^{\text{parent 1}}_{\text{crossover}}$ from the current state (population $\mathcal{S}^{(t)}$);
2. conditioned on the first selected candidate $X^{\text{parent 1}}_{\text{crossover}}$, select the second candidate ligand $X^{\text{parent 2}}_{\text{crossover}}$ from the remaining candidate ligand set $\mathcal{S}^{(t)} - \{X^{\text{parent 1}}_{\text{crossover}}\}$ and apply crossover (Section subsection 9.3.1) to them.

Mutation ($X^{\text{parent}}_{\text{mutation}} \xrightarrow{\text{mutated by } \xi} X^{\text{child}}_{\text{mutation}}$) can be divided to two steps:

1. select the candidate ligand $X^{\text{parent}}_{\text{mutated}}$ to be mutated from the current state (population $\mathcal{S}^{(t)}$);
2. conditioned on the selected candidate ligand $X^{\text{parent}}_{\text{mutated}}$, select the reaction ξ from the reaction set $\mathcal{R}$ and apply it to $X^{\text{parent}}_{\text{mutated}}$.

As applying the crossover and mutation steps are deterministic, the actions in an EMDP focus on selecting parents and mutation types. Upon finishing the action, we could obtain an offspring pool, $\mathcal{Q}^{(t)}$.

State Transition Dynamics. The state transition in an EMDP is identical to the evolution in an evolutionary process. Once we finish the actions and obtain the offspring pool, $\mathcal{Q}^{(t)} = \{X^{\text{child 1}}, X^{\text{child 2}}, \cdots\}$, we apply molecular quality filters to filter out the ones

unlikely to be drug and then select the most promising K to form the parent set for the next generation ($\mathcal{S}^{(t+1)}$).

Reward. We define the reward as the binding affinity change (docking score). The actions leading to stronger binding scores would be prioritized. As there is no "episode" concept in an EMDP, we treat every step equally.

9.3.3 Target-Ligand Policy Network

To utilize molecular structures' translational and rotational invariance, we adopt equivariance neural networks (ENNs) [59] as the target-ligand policy neural networks to select the actions in both mutation and crossover steps. Each ligand has a 3D pose that binds to the target protein, and the complex serves as the input of ENN.

Specifically, we want to model a 3D graph $\mathcal{Y}$, which can be a ligand, target, or target-ligand complex. The input feature can be described as $\mathcal{Y} = (\mathcal{A}, \mathcal{Z})$, where $\mathcal{A}$ represents atoms' categories (the vocabulary set $\mathcal{V} = \{H, C, O, N, \cdots\}$) and $\mathcal{Z}$ represents 3D coordinates of the atoms. Suppose $\mathbf{D} \in \mathbb{R}^{|\mathcal{V}| \times d}$ is the embedding matrix of all the categories of atoms in a vocabulary set $\mathcal{V}$, is randomly initialized and learnable, d is the hidden dimension in ENN. Each kind of atom corresponds to a row in $\mathbf{D}$. We suppose there are N atoms, and each atom corresponds to a node in the 3D graph. Node embeddings at the l-th layer are denoted as $\mathbf{H}^{(l)} = \{\mathbf{h}_i^{(l)}\}_{i=1}^N$, where $l = 0, 1, \cdots, L$, L is number of layers in ENN. The initial node embedding $\mathbf{h}_i^{(0)} = \mathbf{D}^\top \mathbf{a}_i \in \mathbb{R}^d$ embeds the i-th node, where $\mathbf{a}_i$ is one-hot vector that encode the category of the i-th atom. Coordinate embeddings at the l-th layer are denoted $\mathbf{Z}^{(l)} = \{\mathbf{z}_i^{(l)}\}_{i=1}^N$. The initial coordinate embeddings $\mathbf{Z}^{(0)} = \{\mathbf{z}_i\}_{i=1}^N$ are the real 3D coordinates of all the nodes. The following equation defines the feedforward rules of ENN,

for $i, j = 1, \cdots, N,\ i \neq j,\ l = 0, 1, \cdots, L-1$, we have

$$
\begin{aligned}
\mathbf{w}_{ij}^{(l+1)} &= \text{MLP}_e\Big(\mathbf{h}_i^{(l)} \oplus \mathbf{h}_j^{(l)} \oplus ||\mathbf{z}_i^{(l)} - \mathbf{z}_j^{(l)}||_2^2\Big) \ \in \mathbb{R}^d, \\
\mathbf{v}_i^{(l+1)} &= \sum_{j=1, j\neq i}^{N} \mathbf{w}_{ij}^{(l+1)} \ \in \mathbb{R}^d, \\
\mathbf{z}_i^{(l+1)} &= \mathbf{z}_i^{(l)} + \sum_{j=1, j\neq i}^{N} \Big(\mathbf{z}_i^{(l)} - \mathbf{z}_j^{(l)}\Big)\text{MLP}_x\Big(\mathbf{w}_{ij}^{(l)}\Big) \in \mathbb{R}^3, \\
\mathbf{h}_i^{(l+1)} &= \text{MLP}_h\Big(\mathbf{h}_i^{(l)} \oplus \mathbf{v}_i^{(l+1)}\Big) \in \mathbb{R}^d, \\
\mathbf{h}_{\mathcal{Y}} &= \sum_{i=1}^{N} \mathbf{h}_i^{(L)} \ \in \mathbb{R}^d \\
&\implies \underline{\mathbf{h}_{\mathcal{Y}} = \text{ENN}(\mathcal{Y})}
\end{aligned}
\tag{9.1}
$$

where $\oplus$ denotes the concatenation of vectors; $\text{MLP}_e(\cdot) : \mathbb{R}^{2d+1} \to \mathbb{R}^d; \text{MLP}_x(\cdot) : \mathbb{R}^d \to \mathbb{R}; \text{MLP}_h(\cdot) : \mathbb{R}^{2d} \to \mathbb{R}^d$ are all two-layer multiple-layer perceptrons (MLPs) with Swish activation in the hidden layer [96]. At the l-th layer, $\mathbf{w}_{ij}^{(l)}$ represents the message vector for the edge from node i to node j; $\mathbf{v}_i^{(l)}$ represents the message vector for node i, $\mathbf{z}_i^{(l)}$ is the position embedding for node i; $\mathbf{h}_i^{(l)}$ is the node embedding for node i. $\mathbf{H}^{(L)} = [\mathbf{h}_1^{(L)}, \cdots, \mathbf{h}_N^{(L)}]$ are the node embeddings of the L-th (last) layer. We aggregate them using the sum function as a readout function to obtain a representation of the 3D graph, denoted $\mathbf{h}_{\mathcal{Y}}$. The whole process is written as $\mathbf{h}_{\mathcal{Y}} = \text{ENN}(\mathcal{Y})$.

Crossover Policy Network. We design two policy networks for two corresponding actions in a crossover, as mentioned in Section subsection 9.3.2. (1) the first action in crossover operation is to select the first parent ligand $X_{\text{crossover}}^{\text{parent 1}}$ from the population $\mathcal{S}^{(t)}$. Similar to the first action in mutation operation, we obtain a valid probability distribution over all the available ligands based on the target-ligand complex as input feature and ENN as the neural network architecture, the selection probability of the ligand $X_{\text{crossover}}^{\text{parent 1}} \in \mathcal{S}^{(t)}$ is

$$
p_{\text{crossover}}^{(1)}(X_{\text{crossover}}^{\text{parent 1}} | \mathcal{S}^{(t)}) = \frac{\exp\Big(\text{MLP}(\mathbf{h}_{\mathcal{T}\&X_{\text{crossover}}^{\text{parent 1}}})\Big)}{\sum_{X' \in \mathcal{S}^{(t)}} \exp\Big(\text{MLP}(\mathbf{h}_{\mathcal{T}\&X'})\Big)},
\tag{9.2}
$$

where $\mathcal{T}$ and X denotes target and ligand (including 3D pose), respectively, $\mathcal{T}\&X$ denotes target-ligand complex. (2) The second action is to select the second parent ligand conditioned on the first parent ligand selected in the first action. Specifically, for ligand in the remaining population set, we concatenate the ENN's embedding of the target, first parent ligand $X_{\text{crossover}}^{\text{parent 1}}$ and the second parent ligand $X_{\text{mutation}}^{\text{parent 2}}$, and feed it into an MLP to estimate a scalar as an unnormalized probability. The unnormalized probabilities for all the ligands in the remaining population set are normalized via the softmax function, i.e.,

$$\begin{aligned} & p_{\text{crossover}}^{(2)}(X_{\text{crossover}}^{\text{parent 2}} | X_{\text{crossover}}^{\text{parent 1}}, \mathcal{S}^{(t)}) \\ = & \text{Softmax}\big\{\text{MLP}(\mathbf{h}_{\mathcal{T}} \oplus \mathbf{h}_{X_{\text{crossover}}^{\text{parent 1}}} \oplus \mathbf{h}_{X_{\text{crossover}}^{\text{parent 2}}}), \cdots, \big\}_{X_{\text{crossover}}^{\text{parent 2}} \in \mathcal{S}^{(t)} - \{X_{\text{crossover}}^{\text{parent 1}}\}}. \end{aligned} \tag{9.3}$$

Given two parent ligands, crossover finds the largest substructure that the two parent compounds share and generates a child by combining their decorating moieties. Thus, the generation of child ligands is deterministic, and the probability of the generated ligands $X_{\text{crossover}}^{\text{child}}$ is

$$\begin{aligned} & p_{\text{crossover}}(X_{\text{crossover}}^{\text{child 1}}, X_{\text{crossover}}^{\text{child 2}} | \mathcal{S}^{(t)}) \\ = & p_{\text{crossover}}(X_{\text{crossover}}^{\text{parent 1}}, X_{\text{crossover}}^{\text{parent 2}} | \mathcal{S}^{(t)}) \\ = & \, p_{\text{crossover}}^{(1)}(X_{\text{crossover}}^{\text{parent 1}} | \mathcal{S}^{(t)}) \cdot p_{\text{crossover}}^{(2)}(X_{\text{crossover}}^{\text{parent 2}} | X_{\text{crossover}}^{\text{parent 1}}, \mathcal{S}^{(t)}). \end{aligned} \tag{9.4}$$

Mutation Policy Network. We design two policy networks for two corresponding actions in mutation, as mentioned in Section subsection 9.3.2. (1) the first action in the mutation operation is to select a candidate ligand to be mutated from population $\mathcal{S}^{(t)}$. It models the 3D target-ligand complex to learn if there is improvement space in the current complex. Formally, we obtain a valid probability distribution over all the available ligands based on the target-ligand complex as input feature and ENN as neural architecture, the selection

probability of the ligand $X^{\text{parent}}_{\text{mutation}} \in \mathcal{S}^{(t)}$ is

$$p^{(1)}_{\text{mutation}}(X^{\text{parent}}_{\text{mutation}}|\mathcal{S}^{(t)}) = \frac{\exp\big(\text{MLP}(\mathbf{h}_{\mathcal{T}\&X^{\text{parent}}_{\text{mutation}}})\big)}{\sum_{X'\in\mathcal{S}^{(t)}}\exp\big(\text{MLP}(\mathbf{h}_{\mathcal{T}\&X'})\big)}, \tag{9.5}$$

where $\mathcal{T}\&X$ denotes target-ligand complex, $\mathbf{h}_{\mathcal{T}\&X} = \text{ENN}(\mathcal{T}\&X)$ represents the ENN's embedding of the target-ligand complex. (2) The second action is to select the SMARTS reaction from the reaction set conditioned on the selected ligand in the first action. Specifically, for each reaction, we generate the new ligand $X^{\text{child}}_{\text{mutation}}$, then obtain the embedding of the target, first ligand $X^{\text{parent}}_{\text{mutation}}$ and the new ligand $X^{\text{child}}_{\text{mutation}}$ through ENN, concatenate these three embeddings and feed it into an MLP to estimate a scalar as an unnormalized probability. The unnormalized probabilities for all the reactions are normalized via the softmax function, i.e.,

$$p^{(2)}_{\text{mutation}}(\xi|X^{\text{parent}}_{\text{mutation}}, \mathcal{S}^{(t)}) = \text{Softmax}\big\{\text{MLP}(\mathbf{h}_{\mathcal{T}} \oplus \mathbf{h}_{X^{\text{parent}}_{\text{mutation}}} \oplus \mathbf{h}_{X^{\text{child}}_{\text{mutation}}}]), \cdots, \big\}_{\xi\in\mathcal{R}}, \tag{9.6}$$

where $X^{\text{parent}}_{\text{mutation}} \xrightarrow{\text{mutated by } \xi} X^{\text{child}}_{\text{mutation}}$, $\mathcal{R}$ is the reaction set. The probability of the generated ligand $X^{\text{child}}_{\text{mutation}}$ is

$$p_{\text{mutation}}(X^{\text{child}}_{\text{mutation}}|\mathcal{S}^{(t)}) = p^{(1)}_{\text{mutation}}(X^{\text{parent}}_{\text{mutation}}|\mathcal{S}^{(t)}) \cdot p^{(2)}_{\text{mutation}}(\xi|X^{\text{parent}}_{\text{mutation}}, \mathcal{S}^{(t)}). \tag{9.7}$$

Policy Gradient. We leverage policy gradient to train the target-ligand policy neural network. Specifically, we consider maximizing the expected reward as an objective via REINFORCE [100],

$$\max\ \mathbb{E}_{X\sim p(X|\mathcal{S}^{(t)})}\big[\text{Reward}(X)\big], \tag{9.8}$$

where $p(X)$ is defined in Equation (Equation 9.4) and (Equation 9.7) for crossover and mutation mutation, respectively. The reward is the binding affinity between the ligand and the target. Larger reward values are desirable. Each episode corresponds to a generation in RGA and we decompose the whole reward into the sum of multiple intermediate rewards

in multiple generations. The intermediate reward is the improvement of binding affinity over the last generation. The whole pipeline is illustrated in Figure Figure 9.1. To provide a warm start, we pretrain ENN on a 3D target-ligand binding affinity prediction task, where the input is the target-ligand complex and the output is their binding affinity.

9.4 Main Results

In this section, we describe the experimental setup and results.

9.4.1 Experimental Setup

Docking Simulation. We adopt AutoDock Vina [95] to evaluate the binding affinity. The docking score estimated by AutoDock Vina is called the Vina score and roughly characterizes the free energy changes of binding processes in kcal/mol. Thus lower Vina score means a stronger binding affinity between the ligand and target. We picked various disease-related proteins, including G-protein coupling receptors (GPCRs) and kinases from DUD-E [101] and the SARS-CoV-2 main protease [102] as targets.

Baselines. The baseline methods cover traditional brute-force search methods (Screening), deep generative models (JTVAE and Gen3D), genetic algorithm (GA+D, graph-GA, Autogrow 4.0), reinforcement learning methods (MolDQN, RationaleRL, REINVENT, GEGL), and MCMC method (MARS). Gen3D and Autogrow 4.0 are structure-based drug design methods, while others are general-purpose molecular design methods. Although methods explicitly utilizing target structures are relatively few, we add general-purpose molecular design methods optimizing the same docking oracle scores as ours, which is a common use case, as baselines [30, 25]. Concretely, **Screening** mimics high throughput screening via sampling from the ZINC database randomly; **JTVAE** (Junction Tree Variational Auto-Encoder) [11] uses a Bayesian optimization on the latent space to optimize molecules indirectly; **Gen3D** [90] is an auto-regressive generative model that grows 3D structures atom-wise inside the binding pocket; **GA+D** [29] represents molecule as SELF-

IES string [103] and uses genetic algorithm enhanced by a discriminator neural network; **Graph-GA** [30] conduct genetic algorithm on molecular graph representation; **Autogrow 4.0** [93] is the state-of-the-art genetic algorithm in structure-based drug design; **MolDQN** (Molecule Deep Q-Network) [27] leverages deep Q-value learning to grow molecules atom-wisely; **RationaleRL** [28] uses rationale (e.g., functional groups or subgraphs) as the building block and a policy gradient method to guide the training of graph neural network-based generator; **REINVENT** [100] represent molecules as SMILES string and uses policy gradient-based reinforcement learning methods to guide the training of the RNN generator; **GEGL** (genetic expert-guided learning) [104] uses LSTM guided by reinforcement learning to imitate the GA exploration; **MARS** (Markov Molecule Sampling) [65] leverages Markov chain Monte Carlo sampling (MCMC) with the adaptive proposal and annealing scheme to search chemical space. To conduct a fair comparison, we limit the number of oracle calls to 1,000 times for each method. All the baselines can be run with one-line code using the software (https://github.com/wenhao-gao/mol_opt) in practical molecular optimization benchmark [24].

Dataset: we randomly select molecules from ZINC [45] database (around 250 thousand drug-like molecules) as 0-th generation of the genetic algorithms (RGA, Autogrow 4.0, GA+D). ZINC also serves as the training data for pretraining the model in JTVAE, REINVENT, RationaleRL, etc. We adopt CrossDocked2020 [105] dataset that contains around 22 million ligand-protein complexes as the training data for pretraining the policy neural networks, as mentioned in Section subsection 9.3.3.

Metrics. The selection of evaluation metrics follows recent works in molecule optimization [11, 29, 28, 65] and structure-based drug design [93, 90, 25]. For each method, we select top-100 molecules with the best docking scores for evaluation and consider the following metrics: **TOP-1/10/100** (average docking score of top-1/10/100 molecules): docking score directly measures the binding affinity between the ligand and target and is the most informative metric in structure-based drug design; **Novelty (Nov)** (% of the generated molecules that

Table 9.2: The summarized performance of different methods. The mean and standard deviation across targets are reported. Arrows (↑, ↓) indicate the direction of better performance. Screening searches over the existing drug database, ZINC, so the novelty is 0.0%. For each metric, the best method is underlined and the top-3 methods are bolded. RGA-pretrain and RGA-KT are two variants of RGA that are without pretraining and without training on different target proteins, respectively.

Method	TOP-100↓	TOP-10↓	TOP-1↓	Nov↑	Div↑	QED↑	SA↓
screening	-9.351±0.643	-10.433±0.563	-11.400±0.630	0.0±0.0%	0.858±0.005	0.678±0.022	2.689±0.077
MARS	-7.758±0.612	-8.875±0.711	-9.257±0.791	100.0±0.0%	**0.877±0.001**	**0.709±0.008**	**2.450±0.034**
MolDQN	-6.287±0.396	-7.043±0.487	-7.501±0.402	100.0±0.0%	**0.877±0.009**	0.170±0.024	5.833±0.182
GEGL	-9.064±0.920	-9.91±0.990	-10.45±1.040	100.0±0.0%	0.853±0.003	0.643±0.014	2.99±0.054
REINVENT	-10.181±0.441	-11.234±0.632	-12.010±0.833	100.0±0.0%	0.857±0.011	0.445±0.058	2.596±0.116
RationaleRL	-9.233±0.920	-10.834±0.856	-11.642±1.102	100.0±0.0%	0.717±0.025	0.315±0.023	2.919±0.126
JTVAE	-9.291±0.702	-10.242±0.839	-10.963±1.133	98.0±0.027%	0.867±0.001	0.593±0.035	3.222±0.136
Gen3D	-8.686±0.450	-9.285±0.584	-9.832±0.324	100.0±0.0%	**0.870±0.006**	0.701±0.016	3.450±0.120
GA+D	-7.487±0.757	-8.305±0.803	-8.760±0.796	99.2±0.011%	0.834±0.035	0.405±0.024	5.024±0.164
Graph-GA	**-10.848±0.860**	**-11.702±0.930**	**-12.302±1.010**	100.0±0.0%	0.811±0.037	0.456±0.067	3.503±0.367
Autogrow 4.0	**-11.371±0.398**	**-12.213±0.623**	**-12.474±0.839**	100.0±0.0%	0.852±0.011	**0.748±0.022**	**2.497±0.049**
RGA (ours)	**-11.867±0.170**	**-12.564±0.287**	**-12.869±0.473**	100.0±0.0%	0.857±0.020	**0.742±0.036**	**2.473±0.048**
RGA - pretrain	-11.443±0.219	-12.424±0.386	-12.435±0.654	100.0±0.0%	0.854±0.035	0.750±0.034	2.494±0.043
RGA - KT	-11.434±0.169	-12.437±0.354	-12.502±0.603	100.0±0.0%	0.853±0.028	0.738±0.034	2.501±0.050

are not in training set); **Diversity (Div)** (average pairwise Tanimoto distance between the Morgan fingerprints); We also evaluate some simple pharmaceutical properties, including quantitative drug-likeness (**QED**) and synthetic accessibility (**SA**). QED score indicates drug-likeliness ranging from 0 to 1 (the higher the better). SA score ranges from 1 to 10 (the lower the better). All the evaluation functions are available at Therapeutics data commons (TDC, https://tdcommons.ai/fct_overview) [25, 106].

9.4.2 Results

Stronger Optimization Performance. We summarized the main results of the structure-based drug design in Table Table 9.2. We evaluate all the methods on all targets and report each metric's mean and standard deviations across all targets. Our result shows RGA achieves the best performance in TOP-100/10/1 scores among all methods we compared. Compared to Autogrow 4.0, RGA's better performance in docking score demonstrates that the policy networks contribute positively to chemical space navigation and eventually help discover more potent binding molecules. On the other hand, including longer-range navigation steps enabled by crossover leads to superior performance than other RL methods

(REINVENT, MolDQN, GEGL, and RationaleRL) that only focus on local modifications. In addition, we also observed competitive structure quality measured by QED (> 0.7) and SA_Score (< 2.5) in Autogrow 4.0 and RGA without involving them as optimization objectives, thanks to the mutation steps originating from chemical reactions. We visualize two designed ligands with an optimal affinity for closer inspection in Figure Figure 9.2 and Figure 9.3, and find both ligands bind tightly with the targets.

Suppressed Random-Walk Behavior. Especially in SBDD, when the oracle functions are expensive molecular simulations, robustness to random seeds is essential for improving the worst-case performance of algorithms. One of the major issues in traditional GAs is that they have a significant variance between multiple independent runs as they randomly select parents for crossover and mutation types. To examine this behavior, we run five independent runs for RGA, Autogrow 4.0, and graph-GA (three best baselines, all are GA methods) on all targets and plot the standard deviations between runs in Figure Figure 9.4 and Figure 9.5. With policy networks guiding the action steps, we observed that the random-walk behavior in Autogrow 4.0 was suppressed in RGA, indicated by the smaller variance. Especially in the later learning phase (after 500 oracle calls), the policy networks are fine-tuned and guide the search more intelligently. This advantage leads to improved worst-case performance and a higher probability of successfully identifying bioactive drug candidates with constrained resources.

Suppressed Random-Walk Behavior. As mentioned, in the traditional GA, the crossover and mutation operation randomly selects the ligands and reactions; this kind of random-walk behavior usually leads to high variance and is undesirable in molecule optimization. RGA is designed to suppress this random-walk behavior. Specifically, we conduct multiple independent runs to compare the RGA and its random-walk version, i.e., Autogrow 4.0. The average TOP-100 and TOP-10 vina score over 5 independent runs and their standard deviations are reported in Figure Figure 9.4 and Figure 9.5, respectively. We observe that RGA is able to significantly reduce the variance, especially in the later learning phase

(after 500 oracle calls). The observation validates RGA's ability to suppress random-walk behavior.

Knowledge Transfer Between Protein Targets. To verify if RGA benefited from learning the shared physics of ligand-target interaction, we conducted an ablation study whose results are in the last two rows of Table Table 9.2. Specifically, we compare RGA with two variants: (1) RGA-pretrain that does not pretrain the policy network with all native complex structures in the CrossDocked2020; (2) RGA-KT (knowledge transfer) that fine-tune the networks with data of individual target independently. We find that both strategies positively contribute to RGA on TOP-100/10/1 docking score. These results demonstrate the policy networks successfully learn the shared physics of ligand-target interactions and leverage the knowledge to improve their performance.

Knowledge Transfer Between Protein Targets. RGA is able to learn the knowledge of different protein targets using the ENN-based target ligand policy network (Section subsection 9.3.3), which is pretrained on 3D target-ligand binding affinity prediction task. To explore their empirical effect, we compare RGA with two variants: (1) RGA-pretrain does not pretrain the policy network. (2) RGA-KT (knowledge transfer) does not leverage the knowledge learned from other targets and optimize individual target independently. The results are also reported in the last two rows of Table Table 9.2. We find that both strategies positively contribute to RGA on TOP-100/10/1 docking score. The reason is pretraining learns the 3D target-ligand structure information and provides a warm start for the policy network, knowledge transfer (KT) can learn the pattern of target-ligand binding physics from a large amount of data and promote the optimization performance.

Case study. Also, we visualize examples of the binding sites and their top affinity ligands' poses for closer inspection in Figure Figure 9.2 and Figure 9.3. We observe that the ligand binds tightly with the target structure. The example validates the ability of RGA to generate high-binding affinity molecules for the designated target.

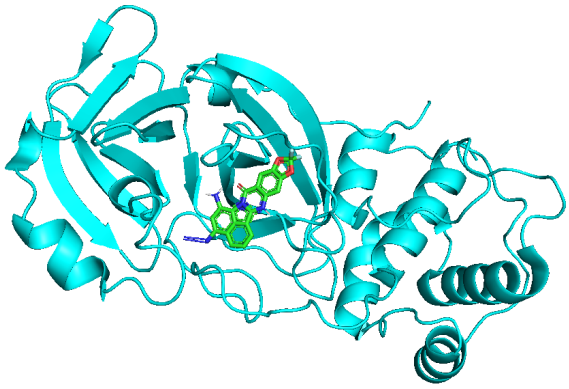


Figure 9.2: Example of ligand poses (generated by RGA) and binding sites of target structures of 7l11.

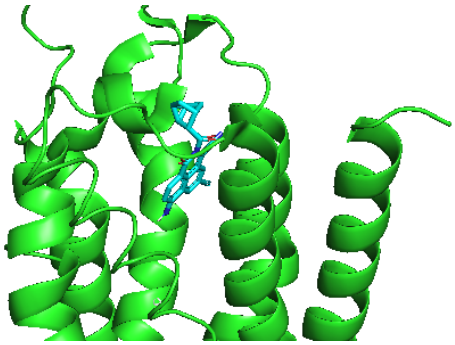


Figure 9.3: Example of ligand poses (generated by RGA) and binding sites of target structures of 3eml.

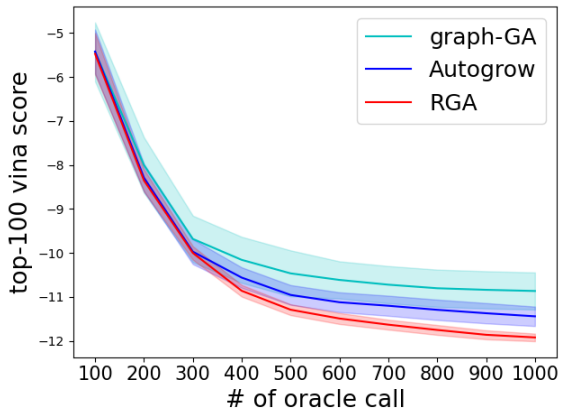


Figure 9.4: Studies of suppressed random-walk behavior. TOP-100 docking score as a function of oracle calls. The results are the means and standard deviations of 5 independent runs.

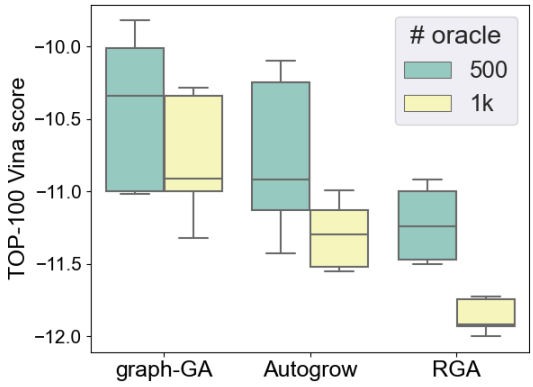


Figure 9.5: The bars of TOP-100 docking score for various independent runs. Studies of suppressed random-walk behavior. TOP-100 docking score as a function of oracle calls. The results are the means and standard deviations of 5 independent runs.

9.5 Conclusion and Discussion

In this work, we propose Reinforced Genetic Algorithm (RGA) to tackle the structure-based drug design problem. RGA reformulates the evolutionary process in genetic algorithms as a Markov decision process called the evolutionary Markov decision process (EMDP) so that the searching processes could benefit from trained neural models. Specifically, we train policy networks to choose the parents to crossover and mutate instead of randomly sampling them. Further, we also leverage the common physics of the ligand-target interaction and adopt a knowledge-transfer strategy that uses data from other targets to train the networks. Through empirical study, we show that RGA has strong and robust optimization performance, consistently outperforming baseline methods in terms of docking score.

Though we adopted mutations originating from chemical reactions and the structural quality metrics seem good, we need to emphasize that the designed molecules from RGA do not guarantee synthesizability [97], as the crossover operations may break inheriting synthesizability. Directly working on synthetic pathways could solve the problem [107, 108], but the extension is not trivial. As for future direction, we expect to analyze the EMDP formulation and the performance of RGA theoretically. We also expect to generalize RGA to other combinatorial optimization scenarios, such as symbolic laws discovery [109], quantum

circuits design [110]), etc.

CHAPTER 10
CONCLUSION AND FUTURE DIRECTIONS

10.1 Conclusion

In summary, my dissertation addresses the fundamental and practical challenges in generative models for drug design. My works contribute novel frameworks and methods that jointly tackle the challenges of efficiency and effectiveness of deep generative models in drug design. Specifically, we focus on the following four generative models:

- **Graph-to-graph model**. The main idea of the graph-to-graph model is to leverage a continuous latent space to represent the discrete drug structure and optimize the latent embedding vectors of molecules.

- **Self-supervised learning** utilizes unlabeled data and predicts a subset of the raw data conditioned on the rest. The conditional probability can be used in generation tasks to update each drug molecule component iteratively and has been applied to both small-molecule drugs (Multi-constraint Molecule Sampling, MIMOSA [15]) and biologics design (sampling method for inverse protein folding (SIPF) [4]).

- **Differentiable programming**. The discrete drug molecules are relaxed to differentiable ones in continuous space, so the gradient of the neural network can be back-propagated to update the differentiable drug molecules directly. The strategy can also be applied to both small-molecule drugs (differentiable scaffolding tree (DST) [5]) and biologics (constrained energy model (CEM) [6]).

- **Intelligent combinatorial optimization**. We enhance the conventional combinatorial optimization methods by using a neural network to prioritize the promising searching branches and suppress the brute-force trial-and-error strategy. We successfully apply

this strategy to the genetic algorithm for small-molecule drug design (Reinforced Genetic Algorithm) [7].

I believe my research advances the frontier of machine learning approaches in the drug discovery process.

10.2 Future Directions

I also describe the following three future directions to extend the completed works from both breadth and depth.

- I expect to build some hybrid generative models to inherit the advantages of multiple categories of generative methods. For example, maximum likelihood learning methods (VAE, GAN, normalizing flow) are good at imitating the known data distribution, while combinatorial optimization-based methods are good at exploring the unknown space. It would be great if we could combine both kinds of methods to get the best of both worlds.

- The comparison between all the generative models is still lacking. I plan to conduct a comprehensive experiment to compare these generative methods systematically. The comprehensive experiment is expected to evaluate the sample efficiency and data efficiency. For a fair comparison, I hope to evaluate all the methods using the same dataset and oracle query budget. Also, I want to thoroughly explore the hyperparameter for each method based on automatic hyperparameter tuning strategies such as Bayesian optimization.

- I plan to build a Python toolkit for deep generative models of drug design, e.g., Jupyter notebook, so that other researchers can easily run the deep generative models for drug design. Concretely, I want to build a unified software environment to standardize the drug design process and build "model card" to record the implementation details for each model to enhance reproducibility and transparency.

REFERENCES

[1] T. Fu, C. Xiao, and J. Sun, "CORE: Automatic molecule optimization using copy and refine strategy," *AAAI*, 2020.

[2] T. Fu, C. Xiao, L. Glass, and J. Sun, "Moler: Incorporate molecule-level reward to enhance deep generative model for molecule optimization," *IEEE TKDE*, 2021.

[3] T. Fu, C. Xiao, X. Li, L. M. Glass, and J. Sun, "Mimosa: Multi-constraint molecule sampling for molecule optimization," *arXiv preprint arXiv:2010.02318*, 2020.

[4] T. Fu and J. Sun, "Sipf: Sampling method for inverse protein folding," in *KDD*, 2022.

[5] T. Fu, W. Gao, C. Xiao, J. Yasonik, C. W. Coley, and J. Sun, "Differentiable scaffolding tree for molecular optimization," *ICLR*, 2022.

[6] T. Fu and J. Sun, "Antibody complementarity determining regions (cdrs) design using constrained energy model," in *KDD*, 2022.

[7] T. Fu, W. Gao, C. W. Coley, and J. Sun, "Reinforced genetic algorithm for structure-based drug design," *NeurIPS*, 2022.

[8] 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*, vol. 27, no. 8, pp. 675–679, 2013.

[9] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *International Conference on Learning Representations*, 2016.

[10] R. Gómez-Bombarelli *et al.*, "Automatic chemical design using a data-driven continuous representation of molecules," *ACS central science*, 2018.

[11] W. Jin, R. Barzilay, and T. Jaakkola, "Junction tree variational autoencoder for molecular graph generation," *ICML*, 2018.

[12] Y. Du, X. Guo, Y. Wang, A. Shehu, and L. Zhao, "Small molecule generation via disentangled representation learning," *Bioinformatics*, btac296, 2022.

[13] N. D. Cao and T. Kipf, *MolGAN: An implicit generative model for small molecular graphs*, 2018. arXiv: 1805.11973.

[14] K. Madhawa, K. Ishiguro, K. Nakago, and M. Abe, "Graphnvp: An invertible flow model for generating molecular graphs," *arXiv preprint arXiv:1905.11600*, 2019.

[15] T. Fu, C. Xiao, X. Li, L. M. Glass, and J. Sun, "MIMOSA: Multi-constraint molecule sampling for molecule optimization," *AAAI*, 2020.

[16] Y. Du, X. Liu, N. Shah, S. Liu, J. Zhang, and B. Zhou, "Chemspace: Interpretable and interactive chemical space exploration," 2022.

[17] N. De Cao and T. Kipf, "Molgan: An implicit generative model for small molecular graphs," *arXiv preprint arXiv:1805.11973*, 2018.

[18] W. Jin, K. Yang, R. Barzilay, and T. Jaakkola, "Learning multimodal graph-to-graph translation for molecular optimization," *ICLR*, 2019.

[19] T. Fu, C. Xiao, L. M. Glass, and J. Sun, "α-mop: Molecule optimization with α-divergence," in *2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, IEEE, 2020, pp. 240–244.

[20] S. Honda, H. Akita, K. Ishiguro, T. Nakanishi, and K. Oono, "Graph residual flow for molecular graph generation," *arXiv preprint arXiv:1909.13521*, 2019.

[21] C. Shi, M. Xu, Z. Zhu, W. Zhang, M. Zhang, and J. Tang, "GraphAF: A flow-based autoregressive model for molecular graph generation," in *ICLR*, 2020.

[22] Y. Du, T. Fu, J. Sun, and S. Liu, "Molgensurvey: A systematic survey in machine learning models for molecule design," *arXiv preprint arXiv:2203.14500*, 2022.

[23] N. Brown, M. Fiscato, M. H. Segler, and A. C. Vaucher, "GuacaMol: Benchmarking models for de novo molecular design," *Journal of chemical information and modeling*, vol. 59, no. 3, pp. 1096–1108, 2019.

[24] W. Gao, T. Fu, J. Sun, and C. W. Coley, "Sample efficiency matters: Benchmarking molecular optimization," *arxiv*, 2022.

[25] K. Huang *et al.*, "Therapeutics data commons: Machine learning datasets and tasks for therapeutics," *NeurIPS Track Datasets and Benchmarks*, 2021.

[26] J. You, B. Liu, R. Ying, V. Pande, and J. Leskovec, "Graph convolutional policy network for goal-directed molecular graph generation," in *NIPS*, 2018.

[27] Z. Zhou, S. Kearnes, L. Li, R. N. Zare, and P. Riley, "Optimization of molecules via deep reinforcement learning," *Scientific reports*, vol. 9, no. 1, pp. 1–10, 2019.

[28] W. Jin, R. Barzilay, and T. Jaakkola, "Multi-objective molecule generation using interpretable substructures," in *International Conference on Machine Learning*, PMLR, 2020, pp. 4849–4859.

[29] A. Nigam, P. Friederich, M. Krenn, and A. Aspuru-Guzik, "Augmenting genetic algorithms with deep neural networks for exploring the chemical space," in *ICLR*, 2020.

[30] J. H. Jensen, "A graph-based genetic algorithm and generative model for the exploration of chemical space," *Chemical science*, 2019.

[31] D. Weininger, "Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules," *Journal of chemical information and computer sciences*, vol. 28, no. 1, pp. 31–36, 1988.

[32] J. Ingraham *et al.*, "Generative models for graph-based protein design," *NeurIPS*, 2019.

[33] Y. Zhang *et al.*, "Prodconn: Protein design using a convolutional neural network," *Proteins: Structure, Function, and Bioinformatics*, 2020.

[34] Y. Qi *et al.*, "Densecpd: Improving the accuracy of neural-network-based computational protein sequence design with densenet," *JCIM*, 2020.

[35] A. Strokach *et al.*, "Fast and flexible protein design using deep graph neural networks," *Cell Systems*, 2020.

[36] Y. Cao *et al.*, "Fold2seq: A joint sequence (1d)-fold (3d) embedding-based generative model for protein design," in *ICML*, 2021.

[37] S. Sinai *et al.*, "Variational auto-encoding of protein sequences," *NeurIPS Computational Biology workshop*, 2017.

[38] Z. Costello and H. G. Martin, "How to hallucinate functional proteins," *arXiv*, 2019.

[39] D. Repecka *et al.*, "Expanding functional protein sequence spaces using generative adversarial networks," *Nature Machine Intelligence*, vol. 3, no. 4, pp. 324–333, 2021.

[40] M. Karimi *et al.*, "De novo protein design for novel folds using wasserstein generative adversarial networks," *JCIM*, 2020.

[41] G. Liu *et al.*, "Antibody complementarity determining region design using high-capacity machine learning," *Bioinformatics*, 2020.

[42] W. Jin, J. Wohlwend, R. Barzilay, and T. Jaakkola, "Iterative refinement graph neural network for antibody sequence-structure co-design," *ICLR*, 2022.

[43] K. Huang, T. Fu, L. M. Glass, M. Zitnik, C. Xiao, and J. Sun, "DeepPurpose: A deep learning library for drug–target interaction prediction," *Bioinformatics*, vol. 36, no. 22-23, pp. 5545–5547, 2020.

[44] A. Vaswani *et al.*, "Attention is all you need," in *NeurIPS*, 2017.

[45] T. Sterling and J. J. Irwin, "ZINC 15–ligand discovery for everyone," *Journal of chemical information and modeling*, vol. 55, no. 11, pp. 2324–2337, 2015.

[46] G. R. Bickerton, G. V. Paolini, J. Besnard, S. Muresan, and A. L. Hopkins, "Quantifying the chemical beauty of drugs," *Nature chemistry*, vol. 4, no. 2, p. 90, 2012.

[47] P. Ertl and A. Schuffenhauer, "Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions," *Journal of cheminformatics*, vol. 1, no. 1, p. 8, 2009.

[48] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[49] D. Rogers and M. Hahn, "Extended-connectivity fingerprints," *Journal of Chemical Information and Modeling*, vol. 50, no. 5, pp. 742–754, 2010.

[50] G. Landrum *et al.*, *RDKit: Open-source cheminformatics*, 2006.

[51] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[52] M. Popova, O. Isayev, and A. Tropsha, "Deep reinforcement learning for de novo drug design," *Sci Adv*, vol. 4, no. 7, eaap7885, Jul. 2018.

[53] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *ICML*, 2016, pp. 1928–1937.

[54] M. J. Kusner, B. Paige, and J. M. Hernández-Lobato, "Grammar variational autoencoder," in *ICML*, 2017, pp. 1945–1954.

[55] H. Dai, Y. Tian, B. Dai, S. Skiena, and L. Song, "Syntax-directed variational autoencoder for structured data," *International Conference on Learning Representations (ICLR)*, 2018.

[56] M. Olivecrona, T. Blaschke, O. Engkvist, and H. Chen, "Molecular de-novo design through deep reinforcement learning," *Journal of cheminformatics*, vol. 9, no. 1, p. 48, 2017.

[57] W. Hu *et al.*, "Strategies for pre-training graph neural networks," in *ICLR*, 2019.

[58] K. Yue and K. A. Dill, "Inverse protein folding problem: Designing polymer sequences.," *Proceedings of the National Academy of Sciences*, 1992.

[59] V. G. Satorras, E. Hoogeboom, and M. Welling, "E(n) equivariant graph neural networks," *International Conference on Machine Learning*, 2021.

[60] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019.*, Association for Computational Linguistics, 2019, pp. 4171–4186.

[61] C. A. Orengo, A. D. Michie, S. Jones, D. T. Jones, M. B. Swindells, and J. M. Thornton, "Cath–a hierarchic classification of protein domain structures," *Structure*, vol. 5, no. 8, pp. 1093–1109, 1997.

[62] A. Leaver-Fay *et al.*, "ROSETTA3: An object-oriented software suite for the simulation and design of macromolecules," in *Methods in enzymology*, vol. 487, Elsevier, 2011, pp. 545–574.

[63] Y. Zhao, Z. Qiao, C. Xiao, L. Glass, and J. Sun, "Pyhealth: A python library for health predictive models," *arXiv preprint arXiv:2101.04209*, 2021.

[64] A. E. Gelfand, "Gibbs sampling," *Journal of the American statistical Association*, vol. 95, no. 452, pp. 1300–1304, 2000.

[65] Y. Xie *et al.*, "MARS: Markov molecular sampling for multi-objective drug discovery," in *ICLR*, 2021.

[66] A. Kulesza and B. Taskar, "Determinantal point processes for machine learning," *arXiv preprint arXiv:1207.6083*, 2012.

[67] L. Chen, G. Zhang, and H. Zhou, "Fast greedy map inference for determinantal point process to improve recommendation diversity," in *Neural Information Processing Systems*, 2018, pp. 5627–5638.

[68] H. Moss, D. Leslie, D. Beck, J. Gonzalez, and P. Rayson, "BOSS: Bayesian optimization over string spaces," *Advances in neural information processing systems*, vol. 33, pp. 15 476–15 486, 2020.

[69] Y. Du *et al.*, "GraphGT: Machine learning datasets for graph generation and transformation," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[70] V. Bagal, R. Aggarwal, P. Vinod, and U. D. Priyakumar, "LigGPT: Molecular generation using a transformer-decoder model," 2021.

[71] K. Korovina *et al.*, "ChemBO: Bayesian optimization of small organic molecules with synthesizable recommendations," in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2020, pp. 3393–3403.

[72] W. Gao, S. P. Mahajan, J. Sulam, and J. J. Gray, "Deep learning in protein structural modeling and design," *Patterns*, p. 100 142, 2020.

[73] K. Saka *et al.*, "Antibody design using lstm based deep generative model from phage display library," *Scientific reports*, vol. 11, no. 1, pp. 1–13, 2021.

[74] Y. Zhang, "I-tasser server for protein 3d structure prediction," *BMC bioinformatics*, 2008.

[75] R. M. MacCallum *et al.*, "Antibody-antigen interactions: Contact analysis and binding site topography," *Journal of molecular biology*, 1996.

[76] I. Sela-Culang, V. Kunik, and Y. Ofran, "The structural basis of antibody-antigen recognition," *Frontiers in immunology*, vol. 4, p. 302, 2013.

[77] A. L. Nelson and J. M. Reichert, "Development trends for therapeutic antibody fragments," *Nature biotechnology*, vol. 27, no. 4, pp. 331–337, 2009.

[78] C. Regep *et al.*, "The h3 loop of antibodies shows unique structural characteristics," *Proteins: Structure, Function*, 2017.

[79] R. L. Stanfield, "Antibody structure," *Microbiology spectrum*, 2014.

[80] M. Brubaker, M. Salzmann, and R. Urtasun, "A family of mcmc methods on implicitly defined manifolds," in *Artificial intelligence and statistics*, PMLR, 2012, pp. 161–172.

[81] R. S. Bohacek, C. McMartin, and W. C. Guida, "The art and practice of structure-based drug design: A molecular modeling perspective," *Medicinal research reviews*, vol. 16, no. 1, pp. 3–50, 1996.

[82] A. Tripathi and V. A. Bankaitis, "Molecular docking: From lock and key to combination lock," *Journal of molecular medicine and clinical applications*, vol. 2, no. 1, 2017.

[83] A. Alon *et al.*, "Structures of the $\sigma 2$ receptor enable docking for bioactive ligand discovery," *Nature*, vol. 600, no. 7890, pp. 759–764, 2021.

[84] J. Jumper *et al.*, "Highly accurate protein structure prediction with AlphaFold," *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[85] M. Varadi *et al.*, "Alphafold protein structure database: Massively expanding the structural coverage of protein-sequence space with high-accuracy models," *Nucleic acids research*, vol. 50, no. D1, pp. D439–D444, 2022.

[86] F. Ren *et al.*, "Alphafold accelerates artificial intelligence powered drug discovery: Efficient discovery of a novel Cyclin-dependent Kinase 20 (CDK20) small molecule inhibitor," *arXiv preprint arXiv:2201.09647*, 2022.

[87] J. Lyu *et al.*, "Ultra-large library docking for discovering new chemotypes," *Nature*, vol. 566, no. 7743, pp. 224–229, 2019.

[88] D. E. Graff, E. I. Shakhnovich, and C. W. Coley, "Accelerating high-throughput virtual screening through molecular pool-based active learning," *Chemical science*, vol. 12, no. 22, pp. 7866–7881, 2021.

[89] F. Gentile *et al.*, "Artificial intelligence–enabled virtual screening of ultra-large chemical libraries with deep docking," *Nature Protocols*, pp. 1–26, 2022.

[90] S. Luo, J. Guan, J. Ma, and J. Peng, "A 3d generative model for structure-based drug design," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[91] Y. Li, J. Pei, and L. Lai, "Structure-based de novo drug design using 3d deep generative models," *Chemical science*, vol. 12, no. 41, pp. 13 664–13 675, 2021.

[92] W. P. Walters and M. Murcko, "Assessing the impact of generative AI on medicinal chemistry," *Nature biotechnology*, vol. 38, no. 2, pp. 143–145, 2020.

[93] J. O. Spiegel and J. D. Durrant, "AutoGrow4: An open-source genetic algorithm for de novo drug design and lead optimization," *Journal of cheminformatics*, vol. 12, no. 1, pp. 1–16, 2020.

[94] A. Tripp, G. N. Simm, and J. M. Hernández-Lobato, "A fresh look at de novo molecular design benchmarks," in *NeurIPS 2021 AI for Science Workshop*, 2021.

[95] O. Trott and A. J. Olson, "Autodock Vina: Improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading," *Journal of computational chemistry*, vol. 31, no. 2, pp. 455–461, 2010.

[96] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *arXiv preprint arXiv:1710.05941*, 2017.

[97] W. Gao and C. W. Coley, "The synthesizability of molecules proposed by generative models," *Journal of chemical information and modeling*, vol. 60, no. 12, pp. 5714–5723, 2020.

[98] J. D. Durrant, S. Lindert, and J. A. McCammon, "AutoGrow 3.0: An improved algorithm for chemically tractable, semi-automated protein inhibitor design," *Journal of Molecular Graphics and Modelling*, vol. 44, pp. 104–112, 2013.

[99] M. Hartenfeller *et al.*, "A collection of robust organic synthesis reactions for in silico molecule design," *Journal of chemical information and modeling*, vol. 51, no. 12, pp. 3093–3098, 2011.

[100] M. Olivecrona, T. Blaschke, O. Engkvist, and H. Chen, "Molecular de-novo design through deep reinforcement learning.," *Journal of Cheminformatics*, 2017.

[101] M. M. Mysinger, M. Carchia, J. J. Irwin, and B. K. Shoichet, "Directory of useful decoys, enhanced (DUD-E): Better ligands and decoys for better benchmarking," *Journal of medicinal chemistry*, vol. 55, no. 14, pp. 6582–6594, 2012.

[102] C.-H. Zhang *et al.*, "Potent noncovalent inhibitors of the main protease of sars-cov-2 from molecular sculpting of the drug perampanel guided by free energy perturbation calculations," *ACS central science*, vol. 7, no. 3, pp. 467–475, 2021.

[103] M. Krenn, F. Häse, A. Nigam, P. Friederich, and A. Aspuru-Guzik, "Self-referencing embedded strings (SELFIES): A 100% robust molecular string representation," *Machine Learning: Science and Technology*, vol. 1, no. 4, p. 045 024, 2020.

[104] S. Ahn, J. Kim, H. Lee, and J. Shin, "Guiding deep molecular optimization with genetic exploration," *Advances in neural information processing systems*, vol. 33, pp. 12 008–12 021, 2020.

[105] P. G. Francoeur *et al.*, "Three-dimensional convolutional neural networks and a cross-docked data set for structure-based drug design," *Journal of Chemical Information and Modeling*, vol. 60, no. 9, pp. 4200–4215, 2020.

[106] K. Huang *et al.*, "Artificial intelligence foundation for therapeutic science," *Nature Chemical Biology*, 2022.

[107] W. Gao, R. Mercado, and C. W. Coley, "Amortized tree generation for bottom-up synthesis planning and synthesizable molecular design," *International Conference on Learning Representations*, 2022.

[108] J. Bradshaw, B. Paige, M. J. Kusner, M. Segler, and J. M. Hernández-Lobato, "Barking up the right tree: An approach to search over molecule synthesis dags," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6852–6866, 2020.

[109] M. Schmidt and H. Lipson, "Distilling free-form natural laws from experimental data," *science*, vol. 324, no. 5923, pp. 81–85, 2009.

[110] Y. Du, T. Huang, S. You, M.-H. Hsieh, and D. Tao, "Quantum circuit architecture search: Error mitigation and trainability enhancement for variational quantum solvers," *arXiv preprint arXiv:2010.10217*, 2020.