

Molecule Optimization via Fragment-based Generative Models

Ziqi Chen¹, Martin Renqiang Min², Srinivasan Parthasarathy^{1,3}, Xia Ning^{1,3,4*}

¹Computer Science and Engineering, The Ohio State University, Columbus, OH 43210

²Machine Learning Department, NEC Labs America, Princeton, NJ 08540

³Translational Data Analytics Institute, The Ohio State University, Columbus, OH 43210

⁴Biomedical Informatics, The Ohio State University, Columbus, OH 43210

chen.8484@buckeyemail.osu.edu, renqiang@nec-labs.com, srini@cse.ohio-state.edu, ning.104@osu.edu

Abstract

In drug discovery, molecule optimization is an important step in order to modify drug candidates into better ones in terms of desired drug properties. With the recent advance of Artificial Intelligence, this traditionally *in vitro* process has been increasingly facilitated by *in silico* approaches. We present an innovative *in silico* approach to computationally optimizing molecules and formulate the problem as to generate optimized molecular graphs via deep generative models. Our generative models follow the key idea of fragment-based drug design, and optimize molecules by modifying their small fragments. Our models learn how to identify the to-be-optimized fragments and how to modify such fragments by learning from the difference of molecules that have good and bad properties. In optimizing a new molecule, our models apply the learned signals to decode optimized fragments at the predicted location of the fragments. We also construct multiple such models into a pipeline such that each of the model in the pipeline is able to optimize one fragment, and thus the entire pipeline is able to modify multiple fragments of molecule if needed. We compare our models with other state-of-the-art methods on benchmark datasets and demonstrate that our methods significantly outperform others with more than 80% property improvement under moderate molecular similarity constraints, and more than 10% property improvement under high molecular similarity constraints.

1 Introduction

Molecule optimization is a critical step in drug discovery to improve desired properties of drug candidates through chemical modification. For example, in lead (molecules showing both activity and selectivity towards a given target) optimization (Jorgensen 2009), the chemical structures of the lead molecules can be altered to improve their selectivity and specificity. Conventionally, such molecule optimization process is planned based on knowledge and experiences from medicinal chemists, and is done via fragment-based screening or synthesis (Verdonk and Hartshorn 2004; de Souza Neto et al. 2020; Hoffer et al. 2018; Gerry and Schreiber 2018). Thus, it is not scalable or automated. Recent *in silico* approaches using deep learning have enabled alternative computationally generative processes to accelerate the conventional paradigm. These deep-learning

methods learn from string-based molecule representations (SMILES) (Sattarov et al. 2019; Sanchez-Lengeling and Aspuru-Guzik 2018) or molecular graphs (Jin, Barzilay, and Jaakkola 2018; You et al. 2018), and generate new ones accordingly (e.g., via connecting atoms and bonds) with better properties. While computationally attractive, these methods do not conform to the *in vitro* molecule optimization process in one very important aspect: molecule optimization needs to retain the majority scaffold of a molecule, but generating entire new molecular structures may not reproduce the scaffold. Therefore, these methods are limited in their potentials to inform and direct *in vitro* molecule optimization.

We propose a novel generative model for molecule optimization that better approximates *in silico* chemical modification. Our method is referred to modifier with one fragment, denoted as Modof¹. Following the idea of fragment-based drug design (Murray and Rees 2009; Hajduk and Greer 2007), Modof predicts a single site of disconnection at a molecule, and modifies the molecule by changing the fragments (e.g., ring systems, linkers, side chains) at that site. Different from existing molecule optimization approaches that encode and decode whole molecular graphs, Modof learns from and encodes the difference between molecules before and after optimization at one disconnection site. To modify a molecule, Modof generates only one fragment that instantiates the expected difference by decoding a sample drawn from its latent space. Then, Modof removes the original fragment at the disconnection site, and attaches the generated fragment at the site. By sampling multiple times, Modof is able to generate multiple optimized candidates. A pipeline of multiple, identical Modof models, denoted as Modof-pipe, is implemented to optimize molecules at multiple disconnection sites in different Modof models iteratively, with the output molecule from one Modof model as the input to the next.

Modof has the following advantages:

- Modof modifies one fragment at a time. It better approximates the *in vitro* chemical modification and retains the majority of molecular scaffolds. Thus, it potentially better informs and directs *in vitro* molecule optimization.
- Modof only encodes and decodes the fragment that needs modification, and allows better modification performance.

*Contact author

¹<https://github.com/ziqi92/Modof>

- Modof-pipe modifies multiple fragments at different disconnection sites iteratively. It enables easier control over and intuitive deciphering of the intermediate modification steps, and therefore better interpretability.
- Modof is a much less complex model. It has at least 83% fewer parameters and uses 26% less training data compared to the state of the art.
- Modof-pipe outperforms the state-of-the-art methods on benchmark datasets, with a 121.0% property improvement without molecular similarity constraints on the optimized molecules, and 82.0% and 10.6% improvement if the optimized molecules need to be at least 0.2 and 0.4 similar to those before optimization, respectively.

2 Related Work

A variety of deep generative models have been developed to generate molecules of desired properties. These generative models include reinforcement learning (RL)-based models, generative adversarial networks (GAN)-based models, flow-based generative models, and variational autoencoder (VAE)-based models, among others. Among RL-based models, You et al. (2018) developed a graph convolutional policy network (GCPN) to sequentially add new atoms and corresponding bonds to construct new molecules; Zhou et al. (2019) developed Molecule Deep Q-Networks (MoldQNN) to employ deep Q-networks to modify the molecules step by step. Cao and Kipf (2018) utilized GANs to directly generate atoms and adjacency tensors of molecular graphs. In the flow-based models, Shi et al. (2020) developed an autoregressive model (GraphAF), in which they learn an invertible mapping between Gaussian distribution and molecule structures, and applied RL to fine tune the generation process. VAE-based generative models are very popular. Jin, Barzilay, and Jaakkola (2018) first decomposed a molecular graph into a junction tree of chemical substructures, and then used a junction tree variational autoencoder (JT-VAE) to generate and assemble new molecules. Jin et al. (2019) developed a junction tree-based encoder-decoder model (JTNN), which learns a translation mapping between a pair of molecules to optimize one into another. Jin, Barzilay, and Jaakkola (2020) replaced the small chemical substructures used in JT-VAE with larger graph motifs, and modified JTNN into an autoregressive hierarchical encoder-decoder model (HierG2G).

The existing methods typically encode the entire molecular graphs, and generate whole, new molecules from an empty or a randomly selected structure. Different from these methods, Modof learns from and encodes the difference between molecules before and after optimization. Thus, the learning and generation processes are less complex, and are able to retain majority molecule scaffolds.

3 Definitions and Notations

3.1 Problem Definition

Following Jin, Barzilay, and Jaakkola (2018), we focus on the optimization of octanol-water partition coefficients (logP) penalized by the synthetic accessibility (Ertl and Schuffenhauer 2009) and ring size. The combined measurement is referred to as *penalized* logP , denoted as plogP .

notation	meaning
$M = (\mathcal{G}, \mathcal{T})$	molecule represented by \mathcal{G} and \mathcal{T}
$\mathcal{G} = (\mathcal{A}, \mathcal{B})$	molecular graph with atoms \mathcal{A} and bonds \mathcal{B}
$\mathcal{T} = (\mathcal{V}, \mathcal{E})$	junction tree with nodes \mathcal{V} and edges \mathcal{E}
a, b_{ij}	an atom and a bond connecting a_i and a_j in \mathcal{G}
n, e_{uv}	a node and an edge connecting n_u and n_v in \mathcal{T}
n_d	site of disconnection
$\mathcal{A}(n), \mathcal{N}(n)$	atoms included in a tree node n , n 's neighbors
\mathbf{x}	atom type embedding
$\mathbf{m}^{(1\dots t)}$	concatenation of $\mathbf{m}^{(1)}, \mathbf{m}^{(2)}, \dots \mathbf{m}^{(t)}$

Table 1: Notations

Higher plogP values indicate higher molecule concentrations in the lipid phase with potentially good synthetic accessibility and simple ring structures. Note that Modof can be used to optimize other properties as well, with the property measurement used as the learning objective.

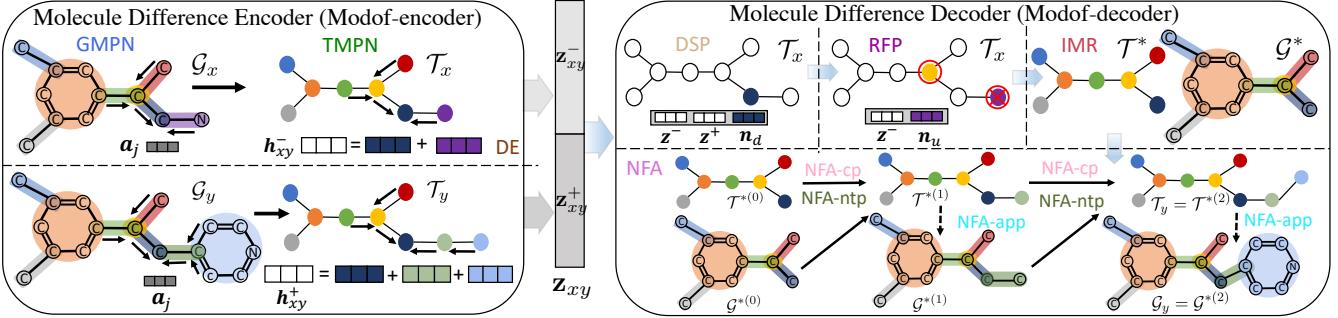
Problem Definition: Given a molecule M_x , molecule optimization aims to optimize M_x into another molecule M_y such that 1) M_y is similar to M_x in its molecular structures (similarity constraint), that is, $\text{sim}(M_x, M_y) > \delta$ (δ is a threshold); and 2) M_y has better properties, that is, $\text{plogP}(M_y) > \text{plogP}(M_x)$ (property constraint).

3.2 Molecule Representations and Notations

We represent a molecule M_x using a molecular graph \mathcal{G}_x and a junction tree \mathcal{T}_x . \mathcal{G}_x is denoted as $\mathcal{G}_x = (\mathcal{A}_x, \mathcal{B}_x)$, where \mathcal{A}_x is the set of atoms in M_x , and \mathcal{B}_x is the set of corresponding bonds. In the junction tree representation $\mathcal{T}_x = (\mathcal{V}_x, \mathcal{E}_x)$ (Jin, Barzilay, and Jaakkola 2018), all the rings and bonds in M_x are extracted as nodes in \mathcal{V}_x ; nodes with common atoms are connected with edges in \mathcal{E}_x . Thus, each node $n \in \mathcal{V}_x$ is a substructure (e.g., a ring, a bond and its connected atoms) in \mathcal{G}_x . We denote the atoms included in node n as $\mathcal{A}_x(n)$ and refer to the nodes connected to n in \mathcal{T}_x as its neighbors, denoted as $\mathcal{N}_x(n)$. Thus, each edge $(n_u, n_v) \in \mathcal{E}_x$ actually corresponds to the common atoms $\mathcal{A}_x(n_u) \cap \mathcal{A}_x(n_v)$ between n_u and n_v . When no ambiguity arises, we will eliminate subscript x in the notations. Note that atoms and bonds are the terms used for molecular graph representations, and nodes and edges are used for junction tree representations. In this paper, all the embedding vectors are by default column vectors, represented by lower-case bold letters; all the matrices are represented by upper-case letters. The key notations are listed in Table 1.

4 Modof Molecule Optimization

Modof modifies one fragment of a molecule (e.g., a ring system, a linker, a side chain) at a time, and thus only encodes and decodes the fragment that needs modification. The site of M where the fragment is modified is referred to as the site of disconnection and denoted as n_d , which corresponds to a node in the junction tree representation. Fig 1 presents an overview of Modof. All the algorithms are presented in Appendix Section A. Discussions on the single-disconnection-site rationale are presented in Appendix Section B.



Modof-encoder: Modof first generates atom embeddings of M_x/M_y over molecular graphs $\mathcal{G}_x/\mathcal{G}_y$ using graph message passing networks (**GMPN**), and node embeddings over corresponding junction tree $\mathcal{T}_x/\mathcal{T}_y$ using tree message passing networks (**TMPN**). The difference between \mathcal{T}_x and \mathcal{T}_y at the disconnection site (\bullet in $\mathcal{T}_x/\mathcal{T}_y$) is encoded (**DE**) into \mathbf{h}_{xy}^- and \mathbf{h}_{xy}^+ , which then construct two normal distributions \mathbf{z}_{xy}^- and \mathbf{z}_{xy}^+ .

Modof-decoder: Using \mathbf{z}_{xy} , Modof conducts disconnection site prediction (**DSP**) to identify the site n_d . At neighbors of n_d , Modof conducts removal fragment prediction (**RFP**) to remove fragment at n_d . Then, Modof produces an intermediate representation (**IMR**) of the remaining scaffold (\mathcal{G}^* , \mathcal{T}^*). Over (\mathcal{G}^* , \mathcal{T}^*), Modof performs new fragment attachment (**NFA**) by interactively performing child node connection prediction (**NFA-cp**), child node type prediction (**NFA-ntp**), and attachment point prediction (**NFA-app**) to optimize M_x .

Molecule representations: substructures in molecular graphs and their corresponding nodes in junction trees are coded in a same color.

Figure 1: Modof Model Overview

4.1 Molecular Difference Encoder (Modof-encoder)

Given two molecules (M_x, M_y), Modof (Algorithm A1 in Appendix) learns and encodes the difference between M_x and M_y using message passing networks (Gilmer et al. 2017) over graphs \mathcal{G}_x and \mathcal{G}_y , denoted as **GMPN**, and over junction trees \mathcal{T}_x and \mathcal{T}_y , denoted as **TMPN**, via three steps.

Step 1. Atom Embedding over Graphs (GMPN) Modof first represents atoms using embeddings to capture atom types and their local neighborhood structures by propagating messages along bonds over molecular graphs. Modof uses an one-hot encoding \mathbf{x}_i to represent the type of atom a_i , and an one-hot encoding \mathbf{x}_{ij} to represent the type of bond b_{ij} connecting a_i and a_j . Each bond b_{ij} is associated with two messages \mathbf{m}_{ij} and \mathbf{m}_{ji} encoding the messages propagating from atom a_i to a_j and vice versa. The $\mathbf{m}_{ij}^{(t)}$ in t -th iteration of GMPN is updated as follows:

$$\mathbf{m}_{ij}^{(t)} = \text{ReLU}(W_1^a \mathbf{x}_i + W_2^a \mathbf{x}_{ij} + W_3^a \sum_{a_k \in \mathcal{N}(a_i) \setminus \{a_j\}} \mathbf{m}_{ki}^{(t-1)}),$$

where $\mathbf{m}_{ki}^{(0)}$ is initialized as zero, and W_i^a 's ($i=1,2,3$) are the learnable parameter matrices. Thus, the message $\mathbf{m}_{ij}^{(t)}$ encodes the information of all length- t paths passing through b_{ij} to a_j in the graph. After t_a iterations of message passing, the atom embedding \mathbf{a}_j is updated as follows:

$$\mathbf{a}_j = \text{ReLU}(U_1^a \mathbf{x}_j + U_2^a \sum_{a_i \in \mathcal{N}(a_j)} \mathbf{m}_{ij}^{(1 \dots t_a)}),$$

where $\mathbf{m}_{ij}^{(1 \dots t_a)}$ is the concatenation of message vectors from all iterations, and U_1^a and U_2^a are learnable parameter matrices. Thus, the atom embedding \mathbf{a}_j aggregates information from a_j 's t_a -hop neighbors, similarly to Xu et al. (2019), to improve the atom embedding representation power.

Step 2. Node Embedding over Junction Trees (TMPN)

Modof encodes nodes in junction trees into embeddings to capture their local neighborhood structures by

passing messages along the tree edges. To produce rich representations of nodes, Modof first aggregates the information of atoms within a node n_u into an embedding \mathbf{s}_u , and the information of atoms shared by a tree edge e_{uv} into an embedding \mathbf{s}_{uv} through the following pooling:

$$\mathbf{s}_u = \sum_{a_i \in \mathcal{A}(n_u)} \mathbf{a}_i, \quad (1) \quad \mathbf{s}_{uv} = \sum_{a_i \in \mathcal{A}(n_u) \cap \mathcal{A}(n_v)} \mathbf{a}_i. \quad (2)$$

Modof also uses a learnable embedding \mathbf{x}_u to represent the type of node n_u . Thus, $\mathbf{m}_{uv}^{(t)}$ from node n_u to n_v in t -th iteration of TMPN is updated as follows:

$$\begin{aligned} \mathbf{m}_{uv}^{(t)} &= \text{ReLU}(W_1^n \text{ReLU}(W_2^n [\mathbf{x}_u; \mathbf{s}_u]) + W_3^n \mathbf{s}_{uv} \\ &\quad + W_4^n \sum_{n_w \in \mathcal{N}(n_u) \setminus \{n_v\}} \mathbf{m}_{wu}^{(t-1)}), \end{aligned}$$

where $[\mathbf{x}_u; \mathbf{s}_u]$ is a concatenation of \mathbf{x}_u and \mathbf{s}_u so as to represent comprehensive node information, and W_i^n 's ($i=1,2,3,4$) are learnable parameter matrices. Similarly to the messages in GMPN, $\mathbf{m}_{uv}^{(t)}$ encodes the information of all length- t paths passing through edge e_{uv} to n_v in the tree. After t_n iterations, the node embedding \mathbf{n}_v is updated as follows:

$$\mathbf{n}_v = \text{ReLU}(U_1^n \text{ReLU}(U_2^n [\mathbf{x}_v; \mathbf{s}_v]) + U_3^n \sum_{n_u \in \mathcal{N}(n_v)} \mathbf{m}_{uv}^{(1 \dots t_n)}), \quad (3)$$

where U_i^n 's ($i=1,2,3$) are the learnable parameter matrices.

Step 3. Difference Embedding (DE) The difference embedding between M_x and M_y is calculated by pooling the node embeddings from \mathcal{T}_x and \mathcal{T}_y as follows:

$$\mathbf{h}_{xy}^- = \sum_{n_x \in \{\mathcal{V}_x \setminus \mathcal{V}_y\} \cup \{n_d \in \mathcal{V}_x\}} \mathbf{n}_x, \quad \mathbf{h}_{xy}^+ = \sum_{n_y \in \{\mathcal{V}_y \setminus \mathcal{V}_x\} \cup \{n_d \in \mathcal{V}_y\}} \mathbf{n}_y,$$

where \mathbf{n}_x 's/ \mathbf{n}_y 's are the embeddings of nodes only appearing in and learned from $\mathcal{T}_x/\mathcal{T}_y$ via TMPN. Note that n_d in the above equations is the site of disconnection, and both \mathcal{T}_x and \mathcal{T}_y have the common node n_d . Thus, \mathbf{h}_{xy}^- essentially represents the fragment that should be removed from M_x at

n_d and \mathbf{h}_{xy}^+ represents the fragment that should be attached to M_x at n_d afterwards in order to modify M_x into M_y . We will discuss how to identify n_d , and the removed and new attached fragments at n_d in M_x and M_y in Section 4.2.

As in variational autoencoder (VAE) (Kingma and Welling 2014), we map the two difference embeddings \mathbf{h}_{xy}^- and \mathbf{h}_{xy}^+ into two normal distributions by computing the mean and log variance with fully connected layers $\mu(\cdot)$ and $\Sigma(\cdot)$. We then sample the latent vectors \mathbf{z}_{xy}^- and \mathbf{z}_{xy}^+ from these two distributions and concatenate them into one latent vector \mathbf{z}_{xy} , that is,

$$\mathbf{z}_{xy}^- \sim N(\mu^-(\mathbf{h}_{xy}^-), \Sigma^-(\mathbf{h}_{xy}^-)), \mathbf{z}_{xy}^+ \sim N(\mu^+(\mathbf{h}_{xy}^+), \Sigma^+(\mathbf{h}_{xy}^+)), \\ \mathbf{z}_{xy} = [\mathbf{z}_{xy}^-; \mathbf{z}_{xy}^+]. \quad (4)$$

Thus, \mathbf{z}_{xy} encodes the difference between M_x and M_y .

4.2 Molecular Difference Decoder (Modof-decoder)

Following the autoencoder idea, Modof decodes the difference embedding \mathbf{z}_{xy} (Eqn 4) into edit operations that change M_x into M_y . Specifically, Modof first predicts a node n_d in \mathcal{T}_x as the disconnection site. This node will split \mathcal{T}_x into several fragments, and the number of the resulted fragments depends on the number of n_d 's neighboring nodes $\mathcal{N}(n_d)$. Modof then predicts which fragments to remove from M_x , and merges the remaining fragments with n_d into an intermediate representation $M^* = (\mathcal{G}^*, \mathcal{T}^*)$. After that, Modof attaches new fragments sequentially starting from n_d to $(\mathcal{G}^*, \mathcal{T}^*)$. The decoding process (Algorithm A2 in Appendix) has the following 4 steps.

Step 1. Disconnection Site Prediction (DSP) Modof predicts a disconnection score for each \mathcal{T}_x 's node n_u as follows,

$$f_d(n_u) = (\mathbf{w}^d)^\top \tanh(W_1^d \mathbf{n}_u + W_2^d \mathbf{z}), \forall n_u \in \mathcal{V}_x, \quad (5)$$

where \mathbf{n}_u is n_u 's embedding (Eqn 3) in \mathcal{T}_x , \mathbf{w}^d and W_i^d 's ($i=1,2$) are learnable parameter vector and matrices, respectively. The node with the largest disconnection score is predicted as the disconnection site n_d . Intuitively, Modof considers the neighboring/local structures of n_u (in \mathbf{n}_u) and “how likely” edit operations (represented by \mathbf{z}) can be applied at n_u . To learn f_d , Modof uses the negative log likelihood of ground-truth disconnection site in tree \mathcal{T}_x as the loss function.

Step 2. Removal Fragment Prediction (RFP) Next, Modof predicts which fragments separated by n_d should be removed from \mathcal{T}_x . For each node n_u connected to n_d , Modof predicts a removal score as follows,

$$f_r(n_u) = \sigma((\mathbf{w}^r)^\top \text{ReLU}(W_1^r \mathbf{n}_u + W_2^r \mathbf{z}^-)), \forall n_u \in \mathcal{V}_x, \quad (6)$$

where $\sigma(\cdot)$ is sigmoid function, \mathbf{w}^r and W_i^r 's ($i=1,2$) are learnable parameter vector and matrices, respectively. The fragment with a removal score greater than 0.5 is predicted to be removed. Thus, there could be multiple or no fragments removed. Intuitively, Modof considers the local structures of the fragment (i.e., \mathbf{n}_u) and “how likely” this fragment should be removed (represented by \mathbf{z}^-). To learn f_r , Modof minimizes binary cross entropy loss to maximize the predicted scores of ground-truth removed fragments in \mathcal{T}_x .

Step 3. Intermediate Representation (IMR) After fragment removal, Modof merges the remaining fragments together with the disconnection site n_d into an intermediate representation $M^* = (\mathcal{G}^*, \mathcal{T}^*)$. M^* may not be a valid molecule after some fragments are removed (some bonds are broken). It represents the scaffold of M_x that should remain unchanged during the optimization. Modof first removes a fragment in order to identify such a scaffold and then adds a fragment to the scaffold to modify the molecule.

Step 4. New Fragment Attachment (NFA) Modof modifies M^* into the optimized M_y by attaching a new fragment (Algorithm A3 in Appendix). Modof uses the following four predictors to sequentially attach new nodes to \mathcal{T}^* . The predictors will be applied iteratively, starting from n_d , on each newly attached node in \mathcal{T}^* . The attached new node in the t -th step is denoted as $n^{*(t)}$ ($n^{*(0)} = n_d$), and the corresponding molecular graph and tree are denoted as $(\mathcal{G}^{*(t)}, \mathcal{T}^{*(t)})$ ($\mathcal{G}^{*(0)} = \mathcal{G}^*$) and $(\mathcal{T}^{*(t)}, \mathcal{T}^{*(0)} = \mathcal{T}^*)$, respectively.

Step 4.1. Child Connection Prediction (NFA-cp) Modof first predicts whether $n^{*(t)}$ should have a new child node attached to it, with the probability calculated as follows:

$$f_c(n^{*(t)}) = \sigma((\mathbf{w}^c)^\top \text{ReLU}(W_1^c \mathbf{n}^{*(t)} + W_2^c \mathbf{z}^+)), \quad (7)$$

where $\mathbf{n}^{*(t)}$ is the embedding of node $n^{*(t)}$ learned over $(\mathcal{T}^{*(t)}, \mathcal{G}^{*(t)})$ (Eqn 3), \mathbf{z}^+ (Eqn 4) indicates “how much” $\mathcal{T}^{*(t)}$ should be expanded, and \mathbf{w}^c and W_i^c 's ($i=1,2$) are learnable parameter vector and matrices. If $f_c(n^{*(t)})$ is above 0.5, Modof predicts that $n^{*(t)}$ should have a new child node and thus child node type prediction will follow; otherwise, the optimization process stops at $n^{*(t)}$. To learn f_c , Modof minimizes a binary cross entropy loss to maximize the probabilities of ground-truth child nodes. Note that $n^{*(t)}$ may have multiple children, and therefore, once a child is generated as in the following steps and attached to $\mathcal{T}^{*(t)}$, another child connection prediction will be conducted at $n^{*(t)}$ with the updated embedding $\mathbf{n}^{*(t)}$ over the expanded $(\mathcal{T}^{*(t)}, \mathcal{G}^{*(t)})$. The above process will be iterated until $n^{*(t)}$ is predicted to have no more children.

Step 4.2. Child Node Type Prediction (NFA-ntp) The new child node of $n^{*(t)}$ is denoted as n_c . Modof predicts the type of n_c by calculating the probabilities of all types of the nodes that can be attached to $n^{*(t)}$ as follows:

$$f_t(n_c) = \text{softmax}(U^l \times \text{ReLU}(W_1^l \mathbf{n}^{*(t)} + W_2^l \mathbf{z}^+)), \quad (8)$$

where $\text{softmax}(\cdot)$ converts a vector of values into probabilities, U^l and W_i^l 's ($i=1,2$) are learnable matrices. Modof assigns the new child n_c the node type x_c corresponding to the highest probability. Modof learns f_t by minimizing cross entropy to maximize the likelihood of true child node types.

Step 4.3. Attachment Point Prediction (NFA-app) If node $n^{*(t)}$ is predicted to have a child node n_c , the next step is to connect $n^{*(t)}$ and n_c . If $n^{*(t)}$ and n_c share one or multiple atoms (e.g., $n^{*(t)}$ and n_c form a fused ring and thus share two adjacent atoms) that can be unambiguously determined as the attachment point(s) based on chemical rules, Modof

will connect $n^{*(t)}$ and n_c via the atom(s). Otherwise, if $n^{*(t)}$ and n_c have multiple connection configurations, Modof predicts the attachment atoms at $n^{*(t)}$ and n_c , respectively.

Step 4.3.1. Attachment Point Prediction at Parent Node (NFA-app-p) Modof scores each candidate attachment point at parent node $n^{*(t)}$, denoted as a_p^* , as follows,

$$g_p(a_p^*) = (\mathbf{w}^p)^\top \tanh(W_1^p \mathbf{a}_p^* + W_2^p \mathbf{x}_c + W_3^p \times \text{ReLU}(U_2^n[\mathbf{x}^{*(t)}; \tilde{\mathbf{s}}^{*(t)}]) + W_4^p \mathbf{z}^+), \quad (9)$$

where $\mathbf{a}_p^* = \sum_{a_i \in a_p^*} \tilde{\mathbf{a}}_i$ represents the embedding of a_p^* (a_p^* could be an atom or a bond), $\tilde{\mathbf{a}}_i$ is calculated by GMPN over $\mathcal{G}^{*(t)}$; U_2^n is as in Eqn 3; $\tilde{\mathbf{s}}^{*(t)}$ is the sum of the embeddings of all atoms in $n^{*(t)}$ (Eqn 1); and \mathbf{w}^p and W_i^p ($i=1,2,3,4$) are learnable vector and matrices. Modof intuitively measures “how likely” a_p^* can be attached to n_c by looking at a_p^* its own (i.e., \mathbf{a}_p^*), its context in $n^{*(t)}$ (i.e., $\mathbf{x}^{*(t)}$) and neighbors $\tilde{\mathbf{s}}^{*(t)}$, its connecting node n_c (i.e., \mathbf{x}_c) and “how much” $n^{*(t)}$ should be expanded (represented by \mathbf{z}^+). The candidate with the highest score is selected as the attachment point in $n^{*(t)}$. Modof learns g_p by minimizing the negative log likelihood of ground-truth attachment points.

Step 4.3.2. Attachment Point Prediction at Child Node (NFA-app-c) Modof scores each candidate attachment point at the child node n_c , denoted as a_c^* , as follows:

$$g_c(a_c^*) = (\mathbf{w}^o)^\top \tanh(W_1^o \mathbf{a}_c^* + W_2^o \mathbf{x}_c + W_3^o \mathbf{a}_p^* + W_4^o \mathbf{z}^+), \quad (10)$$

where $\mathbf{a}_c^* = \sum_{a_i \in a_c^*} \tilde{\mathbf{a}}_i$ represents the embedding of a_c^* (a_c^* could be an atom or a bond) and $\tilde{\mathbf{a}}_i$ is a_i ’s embedding calculated over n_c via GMPN; \mathbf{w}^o and W_i^o ’s ($i=1,2,3,4$) are learnable parameters. Modof intuitively measures “how likely” candidate a_c^* can be attached to a_p^* at $n^{*(t)}$ by looking at a_c^* its own (i.e., \mathbf{a}_c^*), the features of a_p^* (i.e., \mathbf{a}_p^*), its context in n_c (i.e., \mathbf{x}_c) and “how much” $n^{*(t)}$ should be expanded (i.e., \mathbf{z}^+). The candidate with the highest score is selected as the attachment point in n_c . Modof learns g_c by minimizing the negative log likelihood of ground-truth attachment points.

Valence Checking In NFA-app, Modof incorporates valence check to only generate and predict legitimate candidate attachment points that do not violate valence laws.

Sampling Schemes In the decoding process, for each M_x , Modof samples 20 times from the latent space of \mathbf{z} and optimize M_x accordingly. Among all decoded molecules satisfying the similarity constraint with M_x , Modof selects the one of best property as its output.

4.3 Modof Pipeline

A pipeline of Modof models, denoted as Modof-pipe (Algorithm A4 in Appendix), is constructed with a series of identical Modof models, with the output molecule from one Modof model as the input to the next. Given an input molecule $M^{(t)}$ to the t -th Modof model ($M^{(0)}=M$), Modof first optimizes $M^{(t)}$ into $M^{(t+1)}$ as the output of this model. $M^{(t+1)}$ is then fed into the $(t+1)$ -th model if it satisfies the similarity constraint $\text{sim}(M^{(t+1)}, M) > \delta$ and property constraint $\text{plogP}(M^{(t+1)}) > \text{plogP}(M^{(t)})$. Otherwise, $M^{(t)}$ is

output as the final result and Modof-pipe stops. The advantages of this iterative, one-fragment-at-one-time optimization process include that 1) it is easier to control intermediate optimization steps so as to result in optimized molecules of desired similarities and properties; 2) it is easier to optimize multiple fragments in a molecule that are far apart; and 3) it follows a rational molecule design process (Hajduk and Greer 2007) and thus could enable more insights and inform *in vitro* lead optimization.

4.4 Model Training

During model training, we apply teacher forcing to feed the ground truth instead of the prediction results to the sequential decoding process. Following the idea of variational autoencoder, we minimize the following loss function to maximize the likelihood $P(M_y|\mathbf{z}, M_x)$. Thus, the optimization problem is formulated as follows,

$$\min_{\Theta} -\beta D_{\text{KL}}(q_{\phi}(\mathbf{z}|M_x, M_y) \| p_{\theta}(\mathbf{z})) + E_{q_{\phi}(\mathbf{z}|M_x, M_y)} [\log p_{\theta}(M_y|\mathbf{z}, M_x)], \quad (11)$$

where Θ is the set of parameters; $q_{\phi}()$ is an estimated posterior probability function (Modof-encoder); $p_{\theta}(M_y|\mathbf{z}, M_x)$ is the probabilistic decoder representing the likelihood of generating M_y given the latent embedding \mathbf{z} and M_x ; and the prior $p_{\theta}(\mathbf{z})$ follows $\mathcal{N}(\mathbf{0}, \mathbf{I})$. In the above problem, $D_{\text{KL}}()$ is the KL divergence between $q_{\phi}()$ and $p_{\theta}()$. Specifically, the second term represents the prediction/empirical error, defined as the sum of all the loss functions in the above six predictions (Eqn 5-10). We use AMSGRAD (Reddi, Kale, and Kumar 2018) to optimize the learning objective.

5 Experimental Settings

5.1 Data

We use the benchmark training dataset provided by Jin, Barzilay, and Jaakkola (2020). This dataset is extracted from Zinc dataset (Sterling and Irwin 2015) and contains 75K pairs of molecules. Each two paired molecules are similar in their molecule structures but different in their plogP scores. Using DF-GED (Abu-Aisheh et al. 2015) algorithm, we then extract 55,686 pairs of molecules² from Jin’s training dataset such that each extracted pair has only one disconnection site. That is, our training data is 26% less than that in Jin’s. We use these extracted pairs of molecules (104,708 unique molecules) as our training data. Details about the training data generation are available in Section C in Appendix. We use Jin’s validation set for parameter tuning and test on Jin’s testing dataset of 800 molecules. More details are available in Appendix Section D.

5.2 Baseline Methods

We compare Modof with the state-of-the-art baseline methods for the molecule optimization, including JT-VAE (Jin, Barzilay, and Jaakkola 2018), GCPN (You et al. 2018), JTNN (Jin et al. 2019) and HierG2G (Jin, Barzilay, and

²This dataset is available in the /data folder of the repository (<https://github.com/ziqi92/Modof>).

Jaakkola 2020). JT-VAE encodes and decodes junction trees, and assembles new, entire molecular graphs based on decoded junction trees. GCPN applies a graph convolutional policy network and iteratively generates molecules by adding atoms and bonds one by one. JTNN learns from molecule pairs and performs molecule optimization as to translate molecular graphs. HierG2G encodes molecular graphs in a hierarchical fashion, and generates new molecules via generating and connecting structural motifs.

6 Experimental Results

6.1 Overall Comparison

Performance Comparison Table 2 presents the overall comparison between Modof-pipe with a maximum of 5 iterations and the baseline methods. When there is no similarity constraint ($\delta=0$), that is, it is not required to produce similar molecules out of the optimization, Modof-pipe is able to generate highly optimized molecules with substantially better properties (9.28 ± 1.90), with 121.0% improvement from the best baseline GCPN (4.20 ± 1.28), although with lower similarities between the molecules before and after the optimization. However, when the similarity constraint takes effect (e.g., $\delta=0.2$, 0.4 and 0.6), Modof-pipe is able to produce molecules that are both similar to those before optimization and also with better properties. At $\delta=0.2$ and 0.4, Modof-pipe achieves the best property improvement (7.50 ± 1.70 and 4.40 ± 1.27 , respectively) compared to the best baselines (GCPN with 4.12 ± 1.19 at $\delta=0.2$, HierG2G with 3.98 ± 1.47 at $\delta=0.4$), with 82.0% and 10.6% improvement, respectively, though the baselines generate more similar molecules than Modof-pipe. At $\delta=0.6$, Modof-pipe achieves similar performance (2.13 ± 1.00) as the best baselines JTNN and HierG2G (2.33 ± 1.19 and 2.49 ± 1.09 , respectively) and similar molecule similarities. When δ is large, JTNN and HierG2G tend to decode more aromatic rings, leading to large molecules with overestimated similarities. Instead, Modof tends to stop if there are many aromatic rings, and thus, produces more drug-like molecules (Lipinski 2004; Ghose, Viswanadhan, and Wendoloski 1999). Issues related to similarity calculation that will affect optimization performance will be discussed in Appendix Section E. Still, the overall comparison demonstrates that Modof-pipe outperforms or achieves similar performance as the state-of-the-art methods.

Model Complexity The optimal Modof has 28K parameters with $\delta=0.0$, 669K parameters with $\delta=0.2$, and 198K parameters with $\delta=0.4/0.6$, which are far less than those in the baselines. For example, JTNN has 4M parameters and HierG2G has 5M parameters, that is, Modof uses at least 83% fewer parameters and 26% less training data but outperforms or achieves very comparable results as the state-of-the-art baselines. For reproducibility purposes, detailed parameters are reported in Section F of Appendix.

6.2 Pipeline Performance

Overall Performance Table 3 presents the Modof-pipe performance in each of its iterations under $\delta=0.4$. Results for

$\delta=0.0$, 0.2 and 0.6 are presented in Table A3 in Appendix-Section G. Without any similarity constraints (i.e., $\delta=0.0$), Table A3 shows that almost all the 800 testing molecules would be optimized out of each iteration. However, the property improvement (i.e., “ $p_t \pm \text{std}$ ”) gets slower as it becomes more difficult to optimize a good molecule. This is also indicated by the increasing molecule similarities in later iterations (i.e., large $\text{sim}_t \pm \text{std}$ values over iterations). Still, after each iteration, the overall property improvement out of the pipeline (i.e., “ $p \pm \text{std}$ ”) increases (e.g., 9.28 ± 1.90 after iteration 5, same as in Table 2), while the overall molecule similarity (i.e., “ $\text{sim} \pm \text{std}$ ”) decreases over iterations.

With similarity constraint $\delta=0.4$, Table 3 shows very similar trends as to those with $\delta=0.0$. In addition, at $\delta=0.4$, fewer molecules are able to go through further optimizations (“#in%” decreases, “#n%” increases) because the decoded molecules may not satisfy the similarity constraint. Accordingly, the output optimized molecules may not necessarily be of the best properties, and there are a few output molecules with even declined properties (“#n%”). With even higher similarity threshold $\delta=0.6$, Table A3 shows that even fewer molecules (#in%) can be further optimized and property improvement from each iteration is also smaller.

Molecule Similarity Comparison Table 3 also presents the similarities between the optimized molecules and training molecules (i.e., “avgsim w. Trn” and “top-10 sim w. Trn”), and the average pairwise similarities among optimized molecules (i.e., “ $\text{sim}_t M_y$ ”). Table 3 shows that as Modof-pipe optimizes a molecule more, the average pairwise similarities between the optimized and all training molecules (“avgsim w. Trn”) almost remain same with small values. This could be due to the effect that there are many training modules (104,708) and the all-pairwise similarities are smoothed out. However, the average similarities between the optimized molecules and their top-10 most similar training molecules (“top-10 sim w. Trn”) decrease notably. This indicates that Modof generates new molecules that are in general different from training molecules. In addition, the optimized molecules out of each iteration have low similarities around 0.2 (“ $\text{sim}_t M_y$ ”), indicating their diversity. The optimized molecules become slightly more similar to one another. This could indicate that the optimized molecules also share certain similarities due to their good plogP properties (e.g., aromatic proportion (Ritchie and Macdonald 2009)). More results are presented in Table A3 in Appendix.

6.3 Case Study

Fig 2 presents an example of molecule $M_x^{(0)}$ being optimized via 5 iterations of Modof modification into another molecule $M_x^{(5)}$ under $\delta=0.4$. At each iteration, only one, small fragment (highlighted in red in the figure) is modified from its input, and plogP value (below each molecule) is improved. The molecule similarities between $M_x^{(t)}$ ($t=1, 2, \dots, 5$) and $M_x^{(0)}$ are 0.735, 0.581, 0.472, 0.439 and 0.431, respectively. This example also shows that Modof is able to retain the major scaffold of a molecule during the iterative optimization process. Fig 3 presents an example of a molecule that has been optimized at different disconnec-

model	$\delta = 0.0$				$\delta = 0.2$				$\delta = 0.4$				$\delta = 0.6$				
	imprv \pm std		sim \pm std		imprv \pm std		sim \pm std		imprv \pm std		sim \pm std		imprv \pm std		sim \pm std		
JT-VAE	1.91 \pm 2.04	0.28 \pm 0.15	1.68 \pm 1.85	0.33 \pm 0.13	0.84 \pm 1.45	0.51 \pm 0.10	0.21 \pm 0.71	0.69 \pm 0.06									
GCPN	4.20 \pm 1.28	0.32 \pm 0.12	4.12 \pm 1.19	0.34 \pm 0.11	2.49 \pm 1.30	0.47 \pm 0.08	0.79 \pm 0.63	0.68 \pm 0.08									
JTNN	-	-	-	-	3.55 \pm 1.54	0.52	2.33 \pm 1.19	0.67									
HierG2G	-	-	-	-	3.98 \pm 1.46	0.43	2.49 \pm 1.09	0.62									
Modof-pipe	9.28 \pm 1.90	0.16 \pm 0.09	7.50 \pm 1.70	0.24 \pm 0.05	4.40 \pm 1.27	0.43 \pm 0.04	2.13 \pm 1.00	0.62 \pm 0.15									

Columns represent: “imprv”: the average improvement in plogP; “std”: the standard deviation; “sim”: the similarity between the original molecules M_x and optimized molecules M_y ; “-”: not reported in literature.

Table 2: Overall Comparison

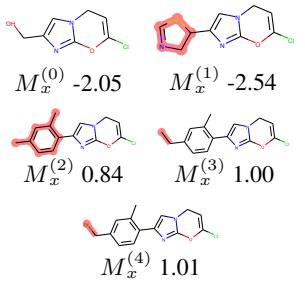


Figure 4: Local Optimization

t	#in%	#p%	#n%	#z%	property improvement			$M_x^{(t)}$ vs $M_y^{(t)}$ sim		avgsim w. Trn		top-10 sim w. Trn		sim _t		
					p \pm std	n \pm std	p \pm std	sim $_t$ \pm std	sim \pm std	all	Trn $_x$	Trn $_y$	all	Trn $_x$	Trn $_y$	M_y
1	100.00	99.75	0.00	0.25	2.79 \pm 1.16	-0.00 \pm 0.00	2.78 \pm 1.16	0.56 \pm 0.10	0.56 \pm 0.10	0.146	0.140	0.152	0.439	0.413	0.416	0.170
2	99.75	84.12	1.50	14.13	1.16 \pm 0.61	-0.27 \pm 0.37	3.75 \pm 1.11	0.74 \pm 0.13	0.46 \pm 0.06	0.151	0.140	0.160	0.430	0.388	0.417	0.188
3	84.12	59.00	2.37	22.75	0.71 \pm 0.53	-0.66 \pm 0.47	4.17 \pm 1.14	0.83 \pm 0.11	0.43 \pm 0.03	0.152	0.139	0.163	0.410	0.366	0.401	0.202
4	59.00	35.50	1.25	22.25	0.46 \pm 0.44	-0.70 \pm 0.61	4.34 \pm 1.21	0.88 \pm 0.09	0.42 \pm 0.02	0.152	0.137	0.165	0.387	0.342	0.381	0.223
5	35.50	18.62	1.25	15.63	0.35 \pm 0.41	-0.35 \pm 0.41	4.40 \pm 1.27	0.91 \pm 0.09	0.41 \pm 0.01	0.149	0.133	0.163	0.366	0.322	0.362	0.237

Columns represent: “t”: the iteration; “#in%”: the number of input molecules in each iteration in percentage over all the testing molecules; “#p”/#n%/#z%”: the percentage of molecules optimized with better/worse/same properties; “p \pm std”: property improvement/decline in the t-th iteration; “p”: the overall property improvement up to the t-th iteration; “sim $_t$ ”/“sim”: the similarities between the molecules before and after optimization in/up to the t-th iteration; “avgsim w. Trn”/“top-10 sim w. Trn”: the average similarities with all/top-10 most similar training molecules; “all”/“Trn $_x$ ”/“Trn $_y$ ”: the comparison molecules identified from all/poor-property/good-property training molecules; “sim $_t$ M_y ”: the average pairwise similarities among optimized molecules.

Table 3: Modof-pipe Performance ($\delta=0.4$)

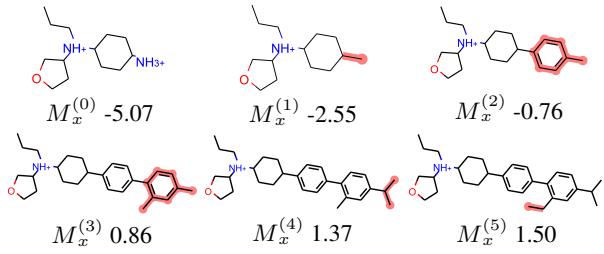


Figure 2: Modof-pipe Optimization Example

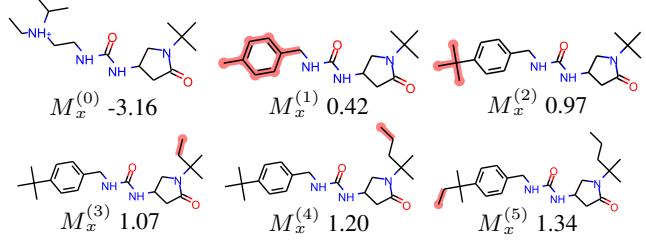


Figure 3: Optimization at Multiple Disconnection Sites

tion sites over Modof-pipe. The molecule similarities between $M_x^{(t)}$ ($t=1,2, \dots, 5$) and $M_x^{(0)}$ are 0.475, 0.471, 0.438, 0.440 and 0.426, respectively. Additional analysis on fragments is available in Appendix Section H.

molecules to Modof will be optimized to the best, and if the optimized molecules do not have better properties, they will not go through additional Modof models. Table 3 shows that at $\delta=0.4$, about 7% of such molecules in total (under “#n%”) stop in the middle of 5-iteration optimization. However, it is possible that such optimized molecules with declined properties might be further optimized in the later Modof models. Fig 4 shows an example of such molecules, where $M_x^{(1)}$ has worse properties after one iteration of optimization, but can be further optimized into $M_x^{(2)}$ of better properties in a later iteration. Note that in our experiments (Table 3), Modof-pipe always stops if no property improvement is observed. Instead, we can exhaustively optimize each molecule through entire Modof-pipe and identify the global optimality.

Conclusions, Limitations and Future Work Modof optimizes molecules at one disconnection site at a time by learning the difference between molecules before and after optimization. With a much less complex model, it achieves significantly better or similar performance compared to the states of the art. In Modof, the modification happens at the periphery of molecules. Although this is very common in *in vitro* lead optimization, we could enrich such optimization and modify the internal regions of molecules, if needed, by learning from proper training data with such regions.

7 Discussions and Conclusions

Local Greedy Optimization Modof-pipe employs a local greedy optimization strategy: in each iteration, the input

References

- Abu-Aisheh, Z.; Raveaux, R.; Ramel, J.-Y.; and Martineau, P. 2015. An Exact Graph Edit Distance Algorithm for Solving Pattern Recognition Problems. In *Proceedings of the International Conference on Pattern Recognition Applications and Methods - Volume 1*, 271–278. Setubal, PRT: Science and Technology Publications, Lda. ISBN 9789897580765. doi:10.5220/0005209202710278.
- Cao, N. D.; and Kipf, T. 2018. MolGAN: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*.
- de Souza Neto, L. R.; Moreira-Filho, J. T.; Neves, B. J.; Maidana, R. L. B. R.; Guimarães, A. C. R.; Furnham, N.; Andrade, C. H.; and Silva Jr, F. P. 2020. In silico strategies to support fragment-to-lead optimization in drug discovery. *Frontiers in Chemistry* 8.
- Ertl, P.; and Schuffenhauer, A. 2009. Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions. *Journal of Cheminformatics* 1(1): 8.
- Gerry, C. J.; and Schreiber, S. L. 2018. Chemical probes and drug leads from advances in synthetic planning and methodology. *Nature Reviews Drug Discovery* 17(5): 333.
- Ghose, A. K.; Viswanadhan, V. N.; and Wendoloski, J. J. 1999. A Knowledge-Based Approach in Designing Combinatorial or Medicinal Chemistry Libraries for Drug Discovery. 1. A Qualitative and Quantitative Characterization of Known Drug Databases. *Journal of Combinatorial Chemistry* 1(1): 55–68. doi:10.1021/cc9800071. PMID: 10746014.
- Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural Message Passing for Quantum Chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, 1263–1272. JMLR.org.
- Hagberg, A.; Swart, P.; and S Chult, D. 2008. Exploring network structure, dynamics, and function using NetworkX. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States).
- Hajduk, P. J.; and Greer, J. 2007. A decade of fragment-based drug design: strategic advances and lessons learned. *Nature Reviews Drug Discovery* 6(3): 211–219.
- Hoffer, L.; Voitovich, Y. V.; Raux, B.; Carrasco, K.; Muller, C.; Fedorov, A. Y.; Derviaux, C.; Amouric, A.; Betzi, S.; Horvath, D.; et al. 2018. Integrated strategy for lead optimization based on fragment growing: the diversity-oriented-target-focused-synthesis approach. *Journal of Medicinal Chemistry* 61(13): 5719–5732.
- Jin, W.; Barzilay, R.; and Jaakkola, T. 2018. Junction Tree Variational Autoencoder for Molecular Graph Generation. volume 80 of *Proceedings of Machine Learning Research*, 2323–2332. Stockholm, Sweden: PMLR.
- Jin, W.; Barzilay, R.; and Jaakkola, T. 2020. Hierarchical Generation of Molecular Graphs using Structural Motifs. *arXiv preprint arXiv:2002.03230*.
- Jin, W.; Yang, K.; Barzilay, R.; and Jaakkola, T. S. 2019. Learning Multimodal Graph-to-Graph Translation for Molecule Optimization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.
- Jorgensen, W. L. 2009. Efficient drug lead discovery and optimization. *Accounts of Chemical Research* 42(6): 724–733.
- Kingma, D. P.; and Welling, M. 2014. Auto-Encoding Variational Bayes. In Bengio, Y.; and LeCun, Y., eds., *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.
- Lipinski, C. A. 2004. Lead-and drug-like compounds: the rule-of-five revolution. *Drug Discovery Today: Technologies* 1(4): 337–341.
- Murray, C.; and Rees, D. 2009. The rise of fragment-based drug discovery. *Nature Chemistry* 1: 187–92. doi:10.1038/nchem.217.
- Reddi, S. J.; Kale, S.; and Kumar, S. 2018. On the Convergence of Adam and Beyond. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.
- Ritchie, T. J.; and Macdonald, S. J. 2009. The impact of aromatic ring count on compound developability – are too many aromatic rings a liability in drug design? *Drug Discovery Today* 14(21): 1011 – 1020. ISSN 1359-6446. doi: https://doi.org/10.1016/j.drudis.2009.07.014.
- Sanchez-Lengeling, B.; and Aspuru-Guzik, A. 2018. Inverse molecular design using machine learning: Generative models for matter engineering. *Science* 361(6400): 360–365. ISSN 0036-8075. doi:10.1126/science.aat2663.
- Sanfeliu, A.; and Fu, K. 1983. A distance measure between attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics SMC-13*(3): 353–362.
- Sattarov, B.; Baskin, I. I.; Horvath, D.; Marcou, G.; Bjerrum, E. J.; and Varnek, A. 2019. De Novo Molecular Design by Combining Deep Autoencoder Recurrent Neural Networks with Generative Topographic Mapping. *Journal of Chemical Information and Modeling* 59(3): 1182–1196. doi:10.1021/acs.jcim.8b00751. PMID: 30785751.
- Shi, C.; Xu, M.; Zhu, Z.; Zhang, W.; Zhang, M.; and Tang, J. 2020. GraphAF: a Flow-based Autoregressive Model for Molecular Graph Generation. In *International Conference on Learning Representations*.
- Sterling, T.; and Irwin, J. J. 2015. ZINC 15-ligand discovery for everyone. *Journal of Chemical Information and Modeling* 55(11): 2324–2337.
- Verdonk, M. L.; and Hartshorn, M. J. 2004. Structure-guided fragment screening for lead discovery. *Current Opinion in Drug Discovery & Development* 7(4): 404.
- Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2019. How Powerful are Graph Neural Networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net.

You, J.; Liu, B.; Ying, Z.; Pande, V.; and Leskovec, J. 2018. Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation. In Bengio, S.; Wallach, H.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 31*, 6410–6421. Curran Associates, Inc.

Zhou, Z.; Kearnes, S.; Li, L.; Zare, R. N.; and Riley, P. 2019. Optimization of molecules via deep reinforcement learning. *Scientific Reports* 9(1): 1–10.

A Algorithms of Modof

Algorithm A1 describes the encoding process of Modof. Modof-encoder takes a pair of molecules as input and encodes the difference of the molecules into vector \mathbf{z}_{xy} . Algorithm A2 describes the decoding process of Modof. Modof-decoder modifies M_x into M_y according to the latent difference vector \mathbf{z}_{xy} . Specifically, in the modification procedure, Modof-decoder first identifies the scaffolds of molecules which should be retained after the optimization, and removes fragments that are not in the scaffolds to get the intermediate representation. Modof-decoder then modifies the intermediate molecule representation M^* into M_y by sequentially attaching new nodes to \mathcal{T}^* in a breadth-first order as in Algorithm A3. Algorithm A4 describes Modof-pipe optimization. Given a molecule M_x , similarity constraint δ , the maximal number of samplings K , and the maximum number of iterations allowed maxIters, our Modof-pipe iteratively optimizes M_x into a new molecule M_y with better property ($\text{plogP}(M_y) > \text{plogP}(M_x)$) under the similarity constraint ($\text{sim}(M_x, M_y) > \delta$).

Algorithm A1 Modof-encoder

Require: $M_x = (\mathcal{G}_x, \mathcal{T}_x)$, $M_y = (\mathcal{G}_y, \mathcal{T}_y)$, n_d

- ▷ Atom embedding
- 1: $\{\mathbf{a}_x\} = \text{GMPN}(\mathcal{G}_x)$
- $\{\mathbf{a}_y\} = \text{GMPN}(\mathcal{G}_y)$
- ▷ Node embedding
- 2: $\{\mathbf{n}_x\} = \text{TMPN}(\mathcal{T}_x, \{\mathbf{a}_x\})$
- $\{\mathbf{n}_y\} = \text{TMPN}(\mathcal{T}_y, \{\mathbf{a}_y\})$
- ▷ Difference embedding
- 3: $\mathbf{z}_{xy} = \text{DE}(\mathcal{T}_x, \{\mathbf{n}_x\}, \mathcal{T}_y, \{\mathbf{n}_y\}, n_d)$
- 4: **return** \mathbf{z}_{xy}

Algorithm A2 Modof-decoder

Require: $M_x = (\mathcal{G}_x, \mathcal{T}_x)$, $\mathbf{z}_{xy} = [\mathbf{z}_{xy}^+, \mathbf{z}_{xy}^-]$

- ▷ Disconnection site prediction
- 1: $n_d = \text{DSP}(\mathcal{T}_x, \mathbf{z}_{xy})$
- ▷ Removal fragment prediction
- 2: $n_r = \text{RFP}(\{\mathbf{n}_u | e_{ub} \in \mathcal{V}_x\}, \mathbf{z}_{xy}^-)$
- ▷ Intermediate representation
- 3: $M^* = \text{IMR}(n_r, M_x)$
- ▷ New fragment attachment
- 4: $M_y = \text{NFA}(M^*, n_d, \mathbf{z}_{xy}^+)$
- 5: **return** M_y

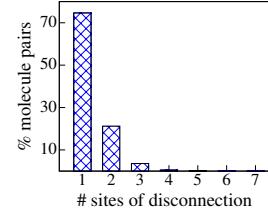
B Disconnection Site Analysis

The key idea of Modof is to modify one fragment at a time under the similarity and property constraints (Section 3.1). This is based on the assumption that for a pair of molecules with a high similarity (e.g., $\text{sim}(M_x, M_y) > 0.6$), they are very likely to have only one (small) fragment different, and

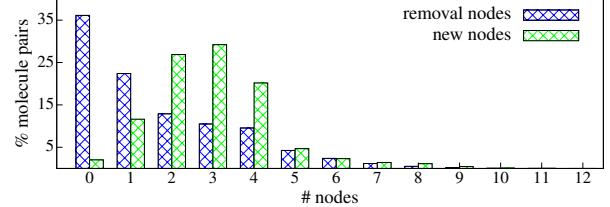
Algorithm A3 Modof New Fragment Attacher NFA

Require: $M^* = (\mathcal{G}^*, \mathcal{T}^*), n_d, \mathbf{z}_{xy}^+$

- 1: $t = 0$
- 2: $M^{*(0)} = M^*$
- 3: $Q = \text{emptyQueue}()$
- 4: $Q.\text{push}(n_d)$
- 5: **while** $Q.\text{isEmpty}()$ **do**
- 6: $n^{*(t)} = Q.\text{pop}()$
 - ▷ Child prediction
- 7: **while** $\text{NFA-cp}(n^{*(t)}, \mathbf{z}_{xy}^+, M^{*(t)})$ **do**
 - ▷ Node type prediction
 - 8: $(n_c, \mathbf{x}_c) = \text{NFA-ntp}(n^{*(t)}, \mathbf{z}_{xy}^+, M^{*(t)})$
 - ▷ Attachment point prediction
 - 9: $(a_p^*, a_c^*) = \text{NFA-app}(n^{*(t)}, n_c, \mathcal{G}^{*(t)}, \mathbf{z}_{xy}^+)$
 - ▷ Attach n_c to $M^{*(t)}$ at $n^{*(t)}$
 - 10: $M^{*(t)} = \text{attach}(M^{*(t)}, n^{*(t)}, a_p^*, n_c, a_c^*)$
 - ▷ Breath-first child expansion
 - 11: $Q.\text{push}(n_c)$
 - 12: **end while**
 - 13: $M^{*(t+1)} = M^{*(t)}$
 - 14: $t = t + 1$
 - 15: **end while**
 - 16: **return** $M^{*(t)}$



(a) # disconnection site distribution



(b) # removal and new node distribution

Figure A1: Distributions of Modification Operations

thus one site of disconnection. Fig A1a presents the number of sites of disconnection among 75K pairs of molecules in the benchmark dataset provided by Jin *et al.* (Jin, Barzilay, and Jaakkola 2020), which each pair has similarity above 0.6. The distribution in Fig A1a shows that when the molecule similarity is high (e.g., 0.6 in the benchmark dataset), most of the molecule pairs (i.e., 74.4%) have only one disconnection site, some of the pairs have two (i.e.,

Algorithm A4 Molecule Optimization via Modof-pipe

Require: $M_x, \delta, K, \text{maxIters}$

1: $M_y^{(0)} = M_x$

2: **for** $t = 1$ to maxIters **do**

▷ input molecule to the t -th Modof module

3: $M_x^{(t)} = M_y^{(t-1)}$

▷ best decoded molecule from this module

4: $M_y^{*(t)} = M_x^{(t)}$

▷ multiple samplings and decoding

5: **for** $k = 0$ to K **do**

6: $\mathbf{z}^{(k)} = \text{sample from } \mathbf{z}$

7: $M = \text{Modof-decoder}(M_x^{(t)}, \mathbf{z}^{(k)})$

▷ the best decoded molecule under constraints

8: **if** $\text{plogP}(M) > \text{plogP}(M_y^{*(t)})$ and
9: $\text{sim}(M, M_x) > \delta$ **then**

10: $M_y^{*(t)} = M$

11: **end if**

12: **end for**

13: **if** $\text{isSame}(M_y^{*(t)}, M_x^{(t)})$ **then**

▷ no more optimization and Modof-pipe stops

14: $t = t - 1$

15: **break**

16: **else**

▷ output molecule from the t -th module

17: $M_y^{(t)} = M_y^{*(t)}$

18: **end if**

19: **end for**

20: $M_y = M_y^{(t)}$

21: **return** M_y

21.1%), and only a few have 3. This indicates that one-fragment modification at a time is a rational idea and directly applicable to the majority of the optimization cases. Even though there could be more disconnection sites, Modof-pipe will still allow multiple-fragment optimization via multiple one-fragment optimizations.

Fig A1b presents the number of nodes that need to be removed from the disconnection sites at M_x , and the number of nodes that need to be attached at the sites afterwards, in Jin’s benchmark dataset (Jin, Barzilay, and Jaakkola 2020). The figures show that on average, more nodes will be added to the disconnection sites compared to the removal nodes. This indicates that the optimized molecules with better properties will become larger, as we have observed in our and other’s methods.

C Training Data Generation

We use a pair of molecules (M_x, M_y) as a training instance in Modof, where M_x and M_y satisfy both the similarity and property constraints, and also M_y is different from M_x in only one fragment at one disconnection site. We construct such training instances as follows. We first quantify the difference between M_x and M_y using the optimal graph edit distance (Sanfeliu and Fu 1983) between \mathcal{T}_x and \mathcal{T}_y , and

derive the optimal edit paths to transform \mathcal{T}_x to \mathcal{T}_y . Such quantification will also identify disconnection sites at M_x during its graph comparison. Details about this process is available in the following Section C.1. Molecule pairs satisfying similarity and property constraints with only one site of disconnection identified will be used as training instances. For a pair of molecules with a high similarity (e.g., > 0.6), it is very likely that they have only one disconnection site as demonstrated in Section B.

C.1 Graph Edit Path Identification

Graph edit distance between tree \mathcal{T}_x and \mathcal{T}_y is defined as the minimum cost to modify \mathcal{T}_x into \mathcal{T}_y with the following graph edit operations:

- Node addition: add a new labeled node into \mathcal{T}_x ;
- Node deletion: delete an existing node from \mathcal{T}_x ;
- Edge addition: add a new edge between a pair of nodes in \mathcal{T}_x ; and
- Edge deletion: delete an existing edge between a pair of nodes in \mathcal{T}_x .

Particularly, we do not allow node or edge substitutions as they can be implemented via deletion and addition operations. We identify the optimal graph edit paths using DF-GED algorithm (Abu-Aisheh et al. 2015) provided by a widely-used package NetworkX (Hagberg, Swart, and S Chult 2008). To identify disconnection sites, we denote the common nodes between \mathcal{V}_x and \mathcal{V}_y as matched nodes \mathcal{M} (i.e., $\mathcal{M} = \mathcal{V}_x \cap \mathcal{V}_y$), nodes only in \mathcal{V}_x as the removal nodes \mathcal{D} (i.e., $\mathcal{D} = \mathcal{V}_x \setminus \mathcal{V}_y$), and the nodes only in \mathcal{V}_y as the new nodes \mathcal{J} (i.e., $\mathcal{J} = \mathcal{V}_y \setminus \mathcal{V}_x$), all with respect to \mathcal{T}_x . Therefore, the disconnection sites will be the matched nodes in \mathcal{T}_x that are also connected with a new node or a removal node, that is, $\{n_d | (n_d \in \mathcal{M}) \wedge (\mathcal{N}(n_d) \cap (\mathcal{D} \cup \mathcal{J}) \neq \emptyset)\}$.

D Data Used in Experiments

Table A1 presents the statistics of the data used in our experiments.

E Similarity Calculation

All baselines except JT-VAE in Table 2 use binary Morgan fingerprints representing the presence or absence of particular, pre-defined substructures. Using such binary fingerprints in molecule similarity calculation may overestimate molecule similarities. An example is presented in Fig A2, where the Tanimoto similarity from binary Morgan fingerprints (sim_b) of the two molecules is 0.644, but they look sufficiently different, with the similarity from Morgan fingerprints of substructure counts (sim_c) only 0.393. According to our experimental results (e.g., Fig 2) and fragment analysis in Section H, we observed that aromatic rings could contribute to large plogP values and thus would be attached to the optimized molecules. Using binary Morgan fingerprints in calculating molecule similarities in this case would easily lead to the solution that many aromatic rings will be attached for molecule optimization, while still satisfying the similarity constraint, but result in very large molecules that

description	value
#training molecules	104,708
#training (M_x, M_y) pairs	55,686
#validation molecules	200
#testing molecules	800
average similarity of training (M_x, M_y) pairs	0.6654
average pairwise similarity between training and testing molecules	0.1193
average training molecule size	25.04
average training $\{M_x\}$ size	22.75
average training $\{M_y\}$ size	27.07
average testing molecule size	20.50
average $\{M_x\}$ plogP	-0.7362
average $\{M_y\}$ plogP	1.1638
average testing molecule plogP	-2.7468
average plogP improvement in training (M_x, M_y) pairs	1.9000

Table A1: Data Statistics

$$\text{sim}_b(M_x, M_y) = 0.644, \quad \text{sim}_c(M_x, M_y) = 0.393$$

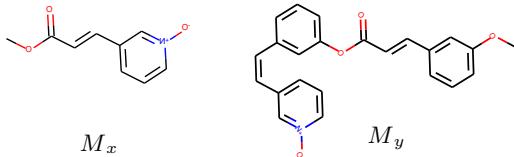


Figure A2: Compound Similarity Comparison

are less drug-like (Ghose, Viswanadhan, and Wendoloski 1999). Therefore, In Modof, we used Morgan fingerprints with substructure counts in calculating molecular similarities (i.e., sim_c).

F Parameters for Reproducibility

Hyper-parameters	Space
hidden layer dimension	{16, 32, 64, 128}
atom/node embedding dimension	{16, 32, 64, 128}
$\mathbf{z}^+/\mathbf{z}^-$ dimension	{8, 16, 32, 64}
# iterations of GMPN	{3, 4, 5, 6}
# iterations of TMPN	{2, 3, 4}
# sampling	20

Table A2: Hyper-Parameter Space

We implemented the models using Python-3.6.9, Pytorch-1.3.1, RDKit-2019.03.4 and NetworkX-2.3. We trained the models on a Tesla P100 GPU and a CPU with 16GB memory with Red Hat Enterprise 7.7. We tuned the hyper-parameters of our models with the grid-search algorithm in the parameter spaces presented in Table A2. We determined the optimal hyper-parameters according to their corresponding

plogP property improvement over the validation molecules. To optimize a molecule, we randomly sample K=20 latent vectors in each iteration.

For $\delta = 0.0$, the optimal dimension of all the hidden layers is 16, and the dimension of latent embedding \mathbf{z} is 16 (i.e., 8 for \mathbf{z}^- and \mathbf{z}^+ , respectively), in Modof. The optimal number of iterations of graph message passing GMPN and tree message passing TMPN is 4. For $\delta = 0.2$, the optimal dimension of all the hidden layers is 128, and the dimension of latent embedding \mathbf{z} is 64, and the optimal iterations of GMPN and TMPN are 4 and 3, respectively. For $\delta = 0.4/0.6$, the dimension of all the hidden layers is 64, the dimension of latent embedding \mathbf{z} is 32, and the iterations of GMPN and TMPN are 5 and 3, respectively.

We optimized the models with learning rate 0.001 and batch size 32. During the training period, we don't use regularization and dropout and use default random number seeds in Pytorch to sample the noise variables employed in the reparameterization trick of VAE. The best performance typically is achieved within 4 epochs of training. We set the KL regularization weight β in the loss function (11) as 0.1 in first epoch, and increased its value by 0.05 every 500 batches until 0.5.

G Pipeline Performance Additional Results

Table A3 presents the Modof-pipe performance in each of its iterations under $\delta = 0.0, 0.2$ and 0.6 . In Modof-pipe, we use up to 5 iterations since they already produce superior performance. Table A4 presents the average size of the optimized molecules in each iteration with $\delta = 0.0, 0.2, 0.4$ and 0.6 . Without any similarity constraints (i.e., $\delta = 0.0$), the average size of the optimized molecules keep increasing as large as 49.56 after 5 iterations of optimization. In addition, the number of added atoms (i.e., $\#a_y - \#a_x$) also becomes larger in later iterations (e.g., for $\delta = 0.0, 1.83, 4.61, 6.57, 7.47, 7.78$ from iteration 1 to 5). This may be due to that in later iterations, Modof identifies fewer fragments which lead to low property scores and should be removed from the input molecules. In the meantime, some general fragments (i.e., aromatic rings) which lead to good property scores are added to the molecules, and thus increase the size of optimized molecules. With similarity constraint $\delta = 0.2, 0.4$ and 0.6 , the size of optimized molecules exhibits similar trends to those with $\delta = 0.0$ but gets smaller. This could be due to that the similarity constraints restrict that the newly added fragments in later iterations should not decrease the similarity values of optimized molecules a lot. This also explain the less property improvement with higher similarity constraint in Table A3.

H Fragment and Molecule Size Analysis

Among training molecule, the top-5 most popular fragments that have been removed from M_x are: O[C:1] (6.43%), N#[C:1] (4.56%), [O-][C:1] (3.23%), [NH3+][C:1] (2.44%), N[C:1]=O (2.17%); the top-5 most popular fragments to be attached into M_y are: CCSSc1cccc[c:1]1 (13.92%), Clc1ccc([C:1])cc1 (12.12%), Clc1c[c:1]ccc1 (6.24%), Clc1cccc[c:1]1 (4.67%), c1ccc2sc([C:1])nc2c1

δ	t	#in%	#p%	#n%	#z%	property improvement			$M_x^{(t)}$ vs $M_y^{(t)}$ sim		avgsim w. Trn			top-10 sim w. Trn			sim _t
						p _t ±std	n _t ±std	p±std	sim _t ±std	sim±std	all	Trn _x	Trn _y	all	Trn _x	Trn _y	M_y
0.0	1	100.00	99.63	0.00	0.38	3.39±1.44	0.00±0.00	3.38±1.45	0.49±0.14	0.49±0.14	0.148	0.141	0.154	0.442	0.415	0.419	0.175
	2	99.63	97.25	0.00	2.38	1.87±0.79	0.00±0.00	5.20±1.61	0.60±0.14	0.31±0.14	0.157	0.142	0.171	0.446	0.392	0.439	0.242
	3	97.25	96.13	0.00	1.13	1.48±0.44	0.00±0.00	6.63±1.67	0.69±0.11	0.24±0.11	0.153	0.133	0.170	0.435	0.355	0.434	0.316
	4	96.13	95.75	0.00	0.38	1.40±0.29	0.00±0.00	7.97±1.77	0.76±0.08	0.19±0.09	0.140	0.119	0.158	0.404	0.316	0.404	0.383
	5	95.75	95.63	0.00	0.13	1.38±0.19	0.00±0.00	9.28±1.90	0.80±0.05	0.16±0.08	0.125	0.105	0.142	0.364	0.280	0.363	0.434
0.2	1	100.00	99.50	0.00	0.50	3.05±1.25	0.00±0.00	3.04±1.27	0.52±0.12	0.52±0.12	0.147	0.142	0.152	0.431	0.415	0.407	0.186
	2	99.50	97.88	0.13	1.50	1.70±0.76	-0.25±0.00	4.70±1.44	0.64±0.11	0.36±0.10	0.150	0.139	0.161	0.405	0.366	0.396	0.242
	3	97.88	93.88	0.50	3.50	1.24±0.45	-0.71±0.29	5.86±1.43	0.74±0.09	0.30±0.08	0.143	0.128	0.155	0.374	0.325	0.371	0.285
	4	93.88	86.13	1.88	5.88	1.08±0.47	-0.69±0.56	6.80±1.52	0.81±0.08	0.26±0.06	0.133	0.118	0.146	0.342	0.294	0.340	0.308
	5	86.13	76.13	3.38	6.63	0.95±0.51	-0.66±0.72	7.50±1.70	0.85±0.06	0.24±0.04	0.125	0.110	0.138	0.315	0.268	0.314	0.322
0.6	1	100.00	95.12	0.63	4.20	1.97±0.84	-0.13±0.12	1.87±0.92	0.68±0.11	0.68±0.11	0.134	0.132	0.137	0.422	0.400	0.393	0.138
	2	95.12	44.37	3.50	47.25	0.48±0.48	-0.31±0.30	2.08±0.97	0.83±0.13	0.65±0.05	0.137	0.133	0.140	0.415	0.393	0.389	0.141
	3	44.37	13.50	1.75	29.12	0.28±0.24	-0.49±0.43	2.12±0.99	0.90±0.10	0.63±0.04	0.145	0.138	0.151	0.404	0.380	0.385	0.162
	4	13.50	2.75	0.63	10.12	0.30±0.23	-0.20±0.24	2.13±1.00	0.93±0.09	0.62±0.03	0.149	0.140	0.156	0.399	0.374	0.383	0.171
	5	2.75	0.50	0.25	2.00	0.17±0.09	-0.03±0.03	2.13±1.00	0.94±0.10	0.62±0.02	0.155	0.144	0.164	0.386	0.360	0.376	0.189

Table A3: Modof-pipe Performance

iter	$\delta = 0.0$		$\delta = 0.2$		$\delta = 0.4$		$\delta = 0.6$	
	# a_x	# a_y						
1	20.51	22.34	20.51	22.51	20.51	22.42	20.51	22.33
2	22.34	26.95	22.53	28.01	22.43	26.23	22.83	23.56
3	27.43	34.00	28.26	34.58	28.01	30.48	27.03	27.31
4	34.20	41.67	35.21	40.95	33.79	35.30	29.82	30.17
5	41.78	49.56	42.29	47.30	38.97	39.81	34.00	34.14

Table A4: Optimized Molecule Size

(4.00%). These removal and attaching fragments are visualized in Fig A3 and Fig A4 in this Appendix.

Overall, the removal fragments are on average of 2.85 atoms and the new attached fragments are of 7.55 atoms. In addition, 39.48% M_x molecules do not have fragments removed and only have new attached fragments, while only 1.78% M_x molecules do not have new fragments attached and only have fragments removed. With $\delta = 0.4/0.6$, the optimized molecules have an average size 29.47/23.41. This shows that in training data, the optimization is typically done via removing small fragments and then attaching larger fragments. This is also reflected in Table A4 (“# a_x ”, “# a_y ”) that out of each Modof iteration, the optimized molecules become larger. We observed the similar trend from JT-VAE and JTNN that their optimized molecules are also larger than those before optimization. In the benchmark data, larger molecules typically have better panelized logP values (e.g., the correlation between molecule size and logP values is 0.42). This indicates that Modof modifies molecules toward better properties.

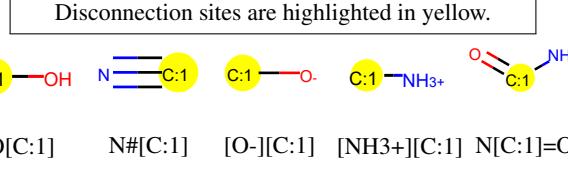


Figure A3: Visualization of Popular Removal Fragments

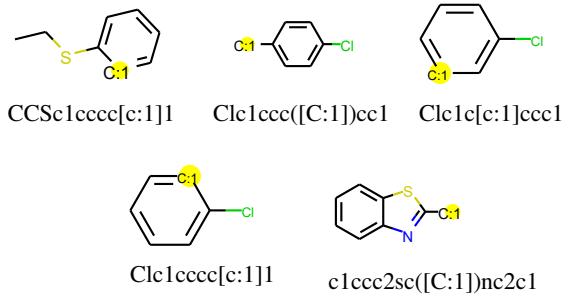


Figure A4: Visualization of Popular Attaching Fragments