Neural Graph Representation Learning with Application to Chemistry

by

Wengong Jin

B.S. in Computer Science
Shanghai Jiao Tong University, 2012

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2018

Author . . . .
Department of Electrical Engineering and Computer Science
May 14, 2018

Certified by . . . .
Regina Barzilay
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by . . . .
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Neural Graph Representation Learning with Application to Chemistry

by

Wengong Jin

Submitted to the Department of Electrical Engineering and Computer Science
on May 14, 2018, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

This thesis focus on deep learning algorithms for learning continuous representation of molecular graphs, a much more compact representation than traditional fingerprints. We demonstrate its better predictive performance in two tasks. First, we seek to automate the prediction of organic reaction outcomes. The previous solution utilizes reaction templates to limit the space, but it suffers from coverage and efficiency issues due to its discrete nature. We propose a template-free approach to efficiently explore the space of product molecules by pinpointing the reaction center. The candidates products are scored by a *Weisfeiler-Lehman Difference Network* that models high-order interactions between changes occurring at nodes across the molecule. Our framework outperforms the top-performing template-based approach with a 10% margin, while running orders of magnitude faster. Moreover, we demonstrate that the model accuracy rivals the performance of domain experts.

Secondly, we seek to automate the design of molecules based on specific chemical properties. Our primary contribution is the direct realization of molecular graphs from continuous space. Our *junction tree variational autoencoder* generates molecular graphs in two phases, by first generating a tree-structured scaffold over chemical substructures, and then combining them into a molecule with a graph message passing network. This approach allows us to incrementally expand molecules while maintaining chemical validity at every step. We evaluate our model on multiple tasks ranging from molecular generation to optimization. Across these tasks, our model outperforms previous state-of-the-art baselines by a significant margin.

Thesis Supervisor: Regina Barzilay
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

First and foremost, I would like to thank my advisor Regina Barzilay and Tommi Jaakkola, whose guidance and assistance was crucial in the completion of this thesis. This work will never be possible without their energy and involvement on the research and their never ending source of support.

I would like to thank those who contributed to the papers on which this thesis is based. First, my sincere thanks to Klavs Jensen, William Green and Connor Coley from MIT chemical engineering department and Tim Jamison from MIT department of chemistry for their invaluable suggestions and discussions during our collaboration. I also thank Tao Lei, Yuan Zhang, Karthik Narasimhan, Jiaming Luo, Tianxiao Shen, Benson Chen, Darsh Shah and every member of MIT NLP group whose ideas, feedback, and assistance were critical to this work. I would also like to acknowledge the support of DARPA Make-It program under contract ARO W911NF-16-2-0023.

Lastly, I am grateful to my mother for her sacrificial love and her support of my study. Her support and love has always been my motivation to overcome many difficulties in my life. I am forever indebted to God and my church family including P. Joseph Han, P. John Peng, Joonhyoon Jeong, Zhenghua Wu, Ce Liu, Hu Huang, Jiaxing Zhang, Donglai Wei, Wenjie Lu, Peilun Dai, and Xun Peng. I have received so much love from my leaders, peers, and most importantly, from God himself who sacrificed his son to save a wretched sinner like me.

Contents

Chapter 1

Introduction

Chemical synthesis planning and drug discovery involves practitioners with years of advanced training and is carried out in a trial-and-error, labor-intensive fashion. For instance, current drug discovery paradigm involves exhaustive screening of large compound libraries against biological targets to find molecules with high potency, which becomes a bottleneck in drug discovery. My goal is to overcome these challenges by deep learning – design novel deep learning approaches to learn chemical knowledge from existing reaction/drug databases. These approaches would potentially automate molecule physiochemical property testing and drug optimization in drug discovery, avoiding exhaustive search in drug design.

On the technical side, my research focus on graph neural network and generative model over graphs, with application to organic reaction prediction and molecule optimization (or drug discovery).

1.1 Reaction Prediction

One of the fundamental problems in organic chemistry is the prediction of which products form as a result of a chemical reaction [52, 54]. While the products can be determined unambiguously for simple reactions, it is a major challenge for many complex organic reactions. Indeed, experimentation remains the primary manner in which reaction outcomes are analyzed. This is time consuming, expensive, and requires the help of an experienced chemist. The empirical approach is particularly limiting for the goal of automatically designing efficient reaction sequences that produce specific target molecule(s), a problem known as chemical retrosynthesis [52, 54].

Viewing molecules as labeled graphs over atoms, we propose to formulate the reaction prediction task as a graph transformation problem. A chemical reaction transforms input molecules (reactants) into new molecules (products) by performing a set of graph edits over reactant molecules, adding new edges and/or eliminating existing ones. Given that a typical reaction may involve more than 100 atoms, fully exploring all possible transformations is intractable. The computational challenge is how to reduce the space of possible edits effectively, and how to select the product from among the resulting candidates.

The state-of-the-art solution is based on *reaction templates* (Figure 1-1). A reaction template specifies a molecular subgraph pattern to which it can be applied and the corresponding graph transformation. Since multiple templates can match a set of reactants, another model is trained to filter candidate products using standard supervised approaches. The key drawbacks of this approach are coverage and scalability. A large number of templates is required to ensure that at least one can reconstitute the correct product. The templates are currently either hand-crafted by experts [19, 5, 50] or generated from reaction databases with heuristic algorithms [6, 32, 9]. For example, Coley et al. [9] extracts 140K unique reaction templates from a database of 1 million reactions. Beyond coverage, applying a template involves graph matching and this makes examining large numbers of templates prohibitively expensive. The current approach is therefore limited to small datasets with limited types of reactions.

In this paper, we propose a template-free approach by learning to identify the *reaction center*, a small set of atoms/bonds that change from reactants to products. In our datasets, on average only 5.5% of the reactant molecules directly participate in the reaction. The small size of the reaction centers together with additional constraints on bond formations enables us to directly enumerate candidate products. Our forward-prediction approach is then divided into two key parts: (1) learning to identify reaction centers and (2) learning to rank the resulting enumerated candidate products.

Our technical approach builds on neural embedding of the *Weisfeiler-Lehman* isomorphism test. We incorporate a specific attention mechanism to identify reaction

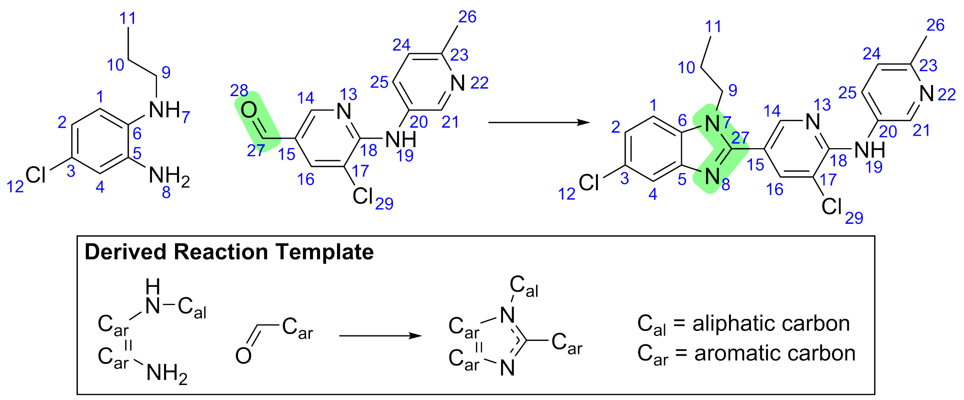


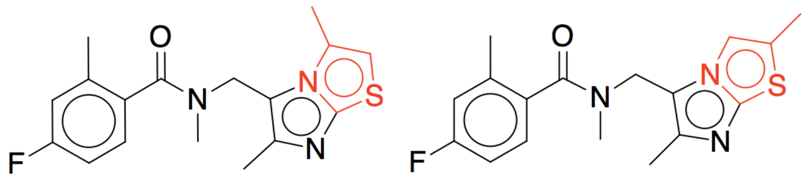
Figure 1-1: An example reaction where the reaction center is (27,28), (7,27), and (8,27), highlighted in green. Here bond (27,28) is deleted and (7,27) and (8,27) are connected by aromatic bonds to form a new ring. The corresponding reaction template consists of not only the reaction center, but nearby functional groups that explicitly specify the context.

centers while leveraging distal chemical effects not accounted for in related convolutional representations [13, 10]. Moreover, we propose a novel *Weisfeiler-Lehman Difference Network* to learn to represent and efficiently rank candidate transformations between reactants and products.

We evaluate our method on two datasets derived from the USPTO [36], and compare our methods to the current top performing system [9]. Our method achieves 83.9% and 77.9% accuracy on two datasets, outperforming the baseline approach by 10%, while running 140 times faster. Finally, we demonstrate that the model outperforms domain experts by a large margin.

1.2 Molecule Optimization

The key challenge of drug discovery is to find target molecules with desired chemical properties. Currently, this task takes years of development and exploration by expert chemists and pharmacologists. Our ultimate goal is to automate this process. From a computational perspective, we decompose the challenge into two complementary subtasks: learning to represent molecules in a continuous manner that facilitates the prediction and optimization of their properties (encoding); and learning to map an optimized continuous representation back into a molecular graph with improved properties (decoding). While deep learning has been extensively investigated for



Cc1cn2c(CN(C)C(=O)c3ccc(F)cc3C)c(C)nc2s1
Cc1cc(F)ccc1C(=O)N(C)Cc1c(C)nc2scc(C)n12

Figure 1-2: Two almost identical molecules with markedly different canonical SMILES in RDKit. The edit distance between two strings is 22 (50.5% of the whole sequence).

molecular graph encoding [13, 25, 15], the harder combinatorial task of molecular graph generation from latent representation remains under-explored.

Prior work on drug design formulated the graph generation task as a string generation problem [16, 30] in an attempt to side-step direct generation of graphs. Specifically, these models start by generating SMILES [56], a linear string notation used in chemistry to describe molecular structures. SMILES strings can be translated into graphs via deterministic mappings (e.g., using RDKit [31]). However, this design has two critical limitations. First, the SMILES representation is not designed to capture molecular similarity. For instance, two molecules with similar chemical structures may be encoded into markedly different SMILES strings (e.g., Figure 1-2). This prevents generative models like variational autoencoders from learning smooth molecular embeddings. Second, essential chemical properties such as molecule validity are easier to express on graphs rather than linear SMILES representations. We hypothesize that operating directly on graphs improves generative modeling of valid chemical structures.

Our primary contribution [23] is a new generative model of molecular graphs. While one could imagine solving the problem in a standard manner – generating graphs node by node – the approach is not ideal for molecules. This is because creating molecules atom by atom would force the model to generate chemically invalid intermediaries (see, e.g., Figure 1-3), delaying validation until a complete graph is generated. Instead, we propose to generate molecular graphs in two phases by exploiting valid subgraphs as components. The overall generative approach, cast as a *junction tree variational autoencoder*, first generates a tree structured object (a junction tree) whose role is to represent the scaffold of subgraph components and their

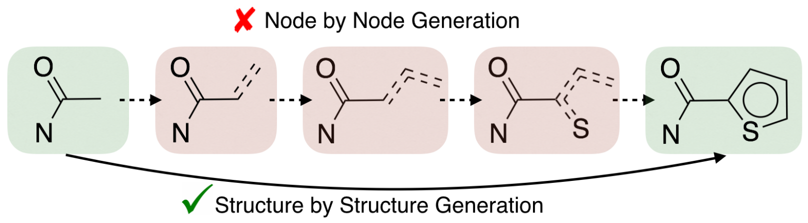


Figure 1-3: Comparison of two graph generation schemes: Structure by structure approach is preferred as it avoids invalid intermediate states (marked in red) encountered in node by node approach.

coarse relative arrangements. The components are valid chemical substructures automatically extracted from the training set using tree decomposition and are used as building blocks. In the second phase, the subgraphs (nodes in the tree) are assembled together into a coherent molecular graph.

We evaluate our model on multiple tasks ranging from molecular generation to optimization of a given molecule according to desired properties. As baselines, we utilize state-of-the-art SMILES-based generation approaches [30, 11]. We demonstrate that our model produces 100% valid molecules when sampled from a prior distribution, outperforming the top performing baseline by a significant margin. In addition, we show that our model excels in discovering molecules with desired properties, yielding a 30% relative gain over the baselines.

Chapter 2

Reaction Prediction

2.1 Overview

Our approach bypasses reaction templates by learning a *reaction center identifier*. Specifically, we train a neural network that operates on the reactant graph to predict a reactivity score for every pair of atoms (Section 2.1.1). A reaction center is then selected by picking a small number of atom pairs with the highest reactivity scores. After identifying the reaction center, we generate possible product candidates by enumerating possible bond configurations between atoms in the reaction center (Section 2.1.2) subject to chemical constraints. We train another neural network to rank these product candidates (represented as graphs, together with the reactants) so that the correct reaction outcome is ranked highest (Section 2.1.3). The overall pipeline is summarized in Figure 2-1. Before describing the two modules in detail, we formally define some key concepts used throughout the paper.

Chemical Reaction A chemical reaction is a pair of molecular graphs (G_r, G_p), where G_r is called the *reactants* and G_p the *products*. A molecular graph is described as $G = (V, E)$, where $V = \{a_1, a_2, \cdots, a_n\}$ is the set of atoms and $E = \{b_1, b_2, \cdots, b_m\}$ is the set of associated bonds of varying types (single, double, aromatic, etc.). Note that G_r is has multiple connected components since there are multiple molecules comprising the reactants. The reactions used for training are *atom-mapped* so that each atom in the product graph has a unique corresponding atom in the reactants.

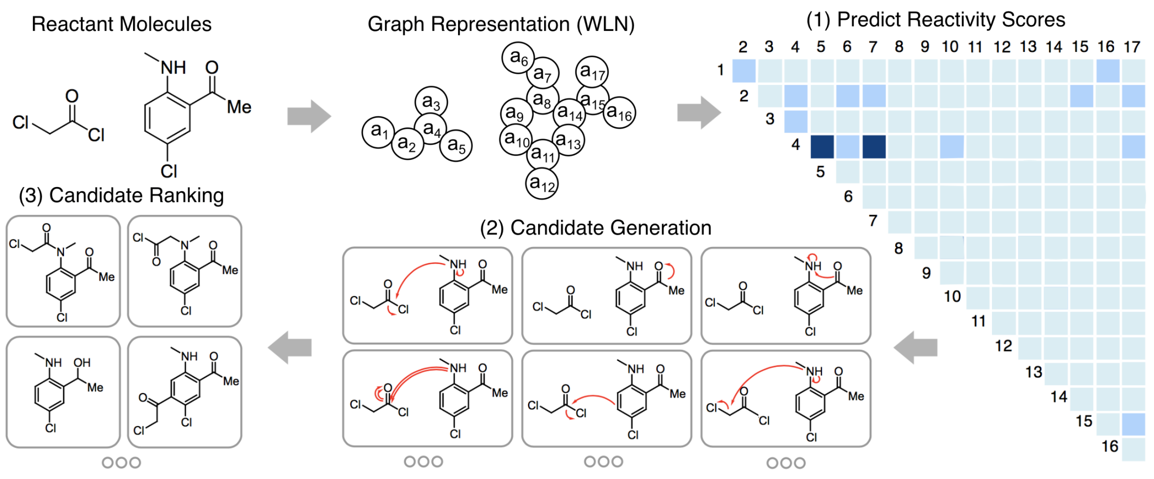


Figure 2-1: Overview of our approach. (1) we train a model to identify pairwise atom interactions in the reaction center. (2) we pick the top K atom pairs and enumerate chemically-feasible bond configurations between these atoms. Each bond configuration generates a candidate outcome of the reaction. (3) Another model is trained to score these candidates to find the true product.

Reaction Center A reaction center is a set of atom pairs $\{(a_i, a_j)\}$, where the bond type between a_i and a_j differs from G_r to G_p. In other words, a reaction center is a minimal set of *graph edits* needed to transform reactants to products. Since the reported reactions in the training set are atom-mapped, reaction centers can be identified automatically given the product.

2.1.1 Reaction Center Identification

In a given reaction $R = (G_r, G_p)$, each atom pair (a_u, a_v) in G_r is associated with a reactivity label $y_{uv} \in \{0, 1\}$ specifying whether their relation differs between reactants and products. The label is determined by comparing G_r and G_p with the help of atom-mapping. We predict the label on the basis of learned atom representations that incorporate contextual cues from the surrounding chemical environment. In particular, we build on a Weisfeiler-Lehman Network (WLN) that has shown superior results against other learned graph representations in the narrower setting of predicting chemical properties of individual molecules [33].

Weisfeiler-Lehman Network (WLN)

The WLN is inspired by the Weisfeiler-Lehman isomorphism test for labeled graphs. The architecture is designed to embed the computations inherent in WL isomorphism testing to generate learned isomorphism-invariant representations for atoms.

WL Isomorphism Test The key idea of the isomorphism test is to repeatedly augment node labels by the sorted set of node labels of neighbor nodes and to compress these augmented labels into new, short labels. The initial labeling is the atom element. In each iteration, its label is augmented with the element labels of its neighbors. Such a multi-set label is compactly represented as a new label by a hash function. Let $c_v^{(L)}$ be the final label of atom a_v. The molecular graph $G = (V, E)$ is represented as a set $\{(c_u^{(L)}, b_{uv}, c_v^{(L)}) \mid (u, v) \in E\}$, where b_{uv} is the bond type between u and v. Two graphs are said to be isomorphic if their set representations are the same. The number of distinct labels grows exponentially with the number of iterations L.

WL Network The discrete relabeling process does not directly generalize to continuous feature vectors. Instead, we appeal to neural networks to continuously embed the computations inherent in the WL test. Let r be the analogous continuous relabeling function. Then a node $v \in G$ with neighbor nodes $N(v)$, node features $\mathbf{f}_v$, and edge features $\mathbf{f}_{uv}$ is "relabeled" according to

$$r(v) = \tau(\mathbf{U}_1\mathbf{f}_v + \mathbf{U}_2 \sum_{u \in N(v)} \tau(\mathbf{V}[\mathbf{f}_u, \mathbf{f}_{uv}])) \tag{2.1}$$

where $\tau(\cdot)$ could be any non-linear function. We apply this relabeling operation iteratively to obtain context-dependent atom vectors

$$\mathbf{h}_v^{(l)} \quad = \quad \tau(\mathbf{U}_1\mathbf{h}_v^{(l-1)} + \mathbf{U}_2 \sum_{u \in N(v)} \tau(\mathbf{V}[\mathbf{h}_u^{(l-1)}, \mathbf{f}_{uv}])) \qquad (1 \leq l \leq L) \tag{2.2}$$

where $\mathbf{h}_v^{(0)} = \mathbf{f}_v$ and $\mathbf{U}_1, \mathbf{U}_2, \mathbf{V}$ are shared across layers. The final atom representations arise from mimicking the set comparison function in the WL isomorphism test, yielding

$$\mathbf{c}_v \quad = \quad \sum_{u \in N(v)} \mathbf{W}^{(0)}\mathbf{h}_u^{(L)} \odot \mathbf{W}^{(1)}\mathbf{f}_{uv} \odot \mathbf{W}^{(2)}\mathbf{h}_v^{(L)} \tag{2.3}$$

The set comparison here is realized by matching each rank-1 edge tensor $\mathbf{h}_u^{(L)} \otimes \mathbf{f}_{uv} \otimes \mathbf{h}_v^{(L)}$ to a set of reference edges also cast as rank-1 tensors $\mathbf{W}^{(0)}[k] \otimes \mathbf{W}^{(1)}[k] \otimes \mathbf{W}^{(2)}[k]$, where $\mathbf{W}[k]$ is the k-th row of matrix $\mathbf{W}$. In other words, Eq. 2.3 above could be written as

$$\mathbf{c}_v[k] = \sum_{u \in N(v)} \left\langle \mathbf{W}^{(0)}[k] \otimes \mathbf{W}^{(1)}[k] \otimes \mathbf{W}^{(2)}[k], \quad \mathbf{h}_u^{(L)} \otimes \mathbf{f}_{uv} \otimes \mathbf{h}_v^{(L)} \right\rangle \tag{2.4}$$

The resulting $\mathbf{c}_v$ is a vector representation that captures the local chemical environment of the atom (through relabeling) and involves a comparison against a learned set of reference environments. The representation of the whole graph G is simply the sum over all the atom representations: $\mathbf{c}_G = \sum_v \mathbf{c}_v$.

Finding Reaction Centers with WLN

We present two models to predict reactivity: the *local* and *global* models. Our local model is based directly on the atom representations $\mathbf{c}_u$ and $\mathbf{c}_v$ in predicting label y_{uv}. The global model, on the other hand, selectively incorporates distal chemical effects with the goal of capturing the fact that atoms outside of the reaction center may be necessary for the reaction to occur. For example, the reaction center may be influenced by certain *reagents*[1]. We incorporate these distal effects into the global model through an attention mechanism.

Local Model Let $\mathbf{c}_u, \mathbf{c}_v$ be the atom representations for atoms u and v, respectively, as returned by the WLN. We predict the reactivity score of (u, v) by passing these through another neural network:

$$s_{uv} = \sigma\left(\mathbf{u}^T \tau(\mathbf{M}_a \mathbf{c}_u + \mathbf{M}_a \mathbf{c}_v + \mathbf{M}_b \mathbf{b}_{uv})\right) \tag{2.5}$$

where $\sigma(\cdot)$ is the sigmoid function, and $\mathbf{b}_{uv}$ is an additional feature vector that encodes auxiliary information about the pair such as whether the two atoms are in different molecules or which type of bond connects them.

Global Model Let α_{uv} be the attention score of atom v on atom u. The global context representation $\tilde{\mathbf{c}}_u$ of atom u is calculated as the weighted sum of all reactant

[1]Molecules that do not typically contribute atoms to the product but are nevertheless necessary for the reaction to proceed.

atoms where the weight comes from the attention module:

$$\begin{aligned}\tilde{\mathbf{c}}_u &= \sum_v \alpha_{uv}\mathbf{c}_v; \qquad \alpha_{uv} = \sigma\left(\mathbf{u}^T \tau(\mathbf{P}_a \mathbf{c}_u + \mathbf{P}_a \mathbf{c}_v + \mathbf{P}_b \mathbf{b}_{uv})\right) && (2.6)\\ s_{uv} &= \sigma\left(\mathbf{u}^T \tau(\mathbf{M}_a \tilde{\mathbf{c}}_u + \mathbf{M}_a \tilde{\mathbf{c}}_v + \mathbf{M}_b \mathbf{b}_{uv})\right) && (2.7)\end{aligned}$$

Note that the attention is obtained with sigmoid rather than softmax non-linearity since there may be multiple atoms relevant to a particular atom u.

Training Both models are trained to minimize the following loss function:

$$\mathcal{L}(\mathcal{T}) = -\sum_{R\in\mathcal{T}} \sum_{u\neq v\in R} y_{uv}\log(s_{uv}) + (1-y_{uv})\log(1-s_{uv}) \tag{2.8}$$

Here we predict each label independently because of the large number of variables. For a given reaction with N atoms, we need to predict the reactivity score of $O(N^2)$ pairs. This quadratic complexity prohibits us from adding higher-order dependencies between different pairs. Nonetheless, we found independent prediction yields sufficiently good performance.

2.1.2 Candidate Generation

We select the top K atom pairs with the highest predicted reactivity score and designate them, collectively, as the reaction center. The set of candidate products are then obtained by enumerating all possible bond configuration changes within the set. While the resulting set of candidate products is exponential in K, many can be ruled out by invoking additional constraints. For example, every atom has a maximum number of neighbors they can connect to (*valence constraint*). We also leverage the statistical bias that reaction centers are very unlikely to consist of disconnected components (*connectivity constraint*). Some multi-step reactions do exist that violate the connectivity constraint. As we will show, the set of candidates arising from this procedure is more compact than those arising from templates without sacrificing coverage.

2.1.3 Candidate Ranking

The training set for candidate ranking consists of lists $\mathcal{T} = \{(r, p_0, p_1, \cdots, p_m)\}$, where r are the reactants, p_0 is the known product, and $p_1, \cdots, p_m$ are other enumerated candidate products. The goal is to learn a scoring function that ranks the highest known product p_0. The challenge in ranking candidate products is again representational. We must learn to represent (r, p) in a manner that can focus on the key difference between the reactants r and products p while also incorporating the necessary chemical contexts surrounding the changes.

We again propose two alternative models to score each candidate pair (r, p). The first model naively represents a reaction by summing difference vectors of all atom representations obtained from a WLN on the associated connected components. Our second and improved model, called WLDN, takes into account higher order interactions between these differences vectors.

WLN with Sum-Pooling Let $\mathbf{c}_v^{(p_i)}$ be the learned atom representation of atom v in candidate product molecule p_i. We define *difference vector* $\mathbf{d}_v^{(p_i)}$ pertaining to atom v as follows:

$$\mathbf{d}_v^{(p_i)} = \mathbf{c}_v^{(p_i)} - \mathbf{c}_v^{(r)}; \qquad s(p_i) = \mathbf{u}^T \tau(\mathbf{M} \sum_{v \in p_i} \mathbf{d}_v^{(p_i)}) \tag{2.9}$$

Recall that the reactants and products are atom-mapped so we can use v to refer to the same atom. The pooling operation is a simple sum over these difference vectors, resulting in a single vector for each (r, p_i) pair. This vector is then fed into another neural network to score the candidate product p_i.

Weisfeiler-Lehman Difference Network (WLDN) Instead of simply summing all difference vectors, the WLDN operates on another graph called a *difference graph*. A difference graph $D(r, p_i)$ is defined as a molecular graph which has the same atoms and bonds as p_i, with atom v's feature vector replaced by $\mathbf{d}_v^{(p_i)}$. Operating on the difference graph has several benefits. First, in $D(r, p_i)$, atom v's feature vector deviates from zero only if it is close to the reaction center, thus focusing the processing on the reaction center and its immediate context. Second, $D(r, p_i)$ explicates neighbor dependencies between difference vectors. The WLDN maps this graph-based representation into a fixed-length vector, by applying a separately parameterized WLN on top of $D(r, p_i)$:

$$\mathbf{h}_v^{(p_i,l)} = \tau\left(\mathbf{U}_1\mathbf{h}_v^{(p_i,l-1)} + \mathbf{U}_2\sum_{u\in N(v)}\tau\left(\mathbf{V}[\mathbf{h}_u^{(p_i,l-1)}, \mathbf{f}_{uv}]\right)\right) \quad (1 \le l \le L) \tag{2.10}$$

$$\mathbf{d}_v^{(p_i,L)} = \sum_{u\in N(v)} \mathbf{W}^{(0)}\mathbf{h}_u^{(p_i,L)} \odot \mathbf{W}^{(1)}\mathbf{f}_{uv} \odot \mathbf{W}^{(2)}\mathbf{h}_v^{(p_i,L)} \tag{2.11}$$

where $\mathbf{h}_v^{(p_i,0)} = \mathbf{d}_v^{(p_i)}$. The final score of p_i is $s(p_i) = \mathbf{u}^T\tau(\mathbf{M}\sum_{v\in p_i}\mathbf{d}_v^{(p_i,L)})$.

Training Both models are trained to minimize the softmax log-likelihood objective over the scores $\{s(p_0), s(p_1), \cdots, s(p_m)\}$ where $s(p_0)$ corresponds to the target.

2.2 Experiments

Data As a source of data for our experiments, we used reactions from USPTO granted patents, collected by Lowe [36]. After removing duplicates and erroneous reactions, we obtained a set of 480K reactions, to which we refer in the paper as USPTO. This dataset is divided into 400K, 40K, and 40K for training, development, and testing purposes.[2]

In addition, for comparison purposes we report the results on the subset of 15K reaction from this dataset (referred as USPTO-15K) used by Coley et al. [9]. They selected this subset to include reactions covered by the 1.7K most common templates. We follow their split, with 10.5K, 1.5K, and 3K for training, development, and testing.

Setup for Reaction Center Identification The output of this component consists of K atom pairs with the highest reactivity scores. We compute the *coverage* as the proportion of reactions where all atom pairs in the true reaction center are predicted by the model, i.e., where the recorded product is found in the model-generated candidate set.

The model features reflect basic chemical properties of atoms and bonds. Atom-level features include its elemental identity, degree of connectivity, number of attached hydrogen atoms, implicit valence, and aromaticity. Bond-level features include bond type (single, double, triple, or aromatic), whether it is conjugated, and whether the bond is part of a ring.

[2] Code and data available at https://github.com/wengong-jin/nips17-rexgen

Both our local and global models are build upon a Weisfeiler-Lehman Network, with unrolled depth 3. All models are optimized with Adam [26], with learning rate decay factor 0.9.

Setup for Candidate Ranking The goal of this evaluation is to determine whether the model can select the correct product from a set of candidates derived from reaction center. We first compare model accuracy against the top-performing template-based approach by Coley et al. [9]. This approach employs frequency-based heuristics to construct reaction templates and then uses a neural model to rank the derived candidates. As explained above, due to the scalability issues associated with this baseline, we can only compare on USPTO-15K, which the authors restricted to contain only examples that were instantiated by their most popular templates. For this experiment, we set $K = 8$ for candidate generation, which achieves 90% coverage and yields 250 candidates per reaction. To compare a standard WLN representation against its counterpart with Difference Networks (WLDN), we train them under the same setup on USPTO-15K, fixing the number of parameters to 650K.

Next, we evaluate our model on USPTO for large scale evaluation. We set $K = 6$ for candidate generation and report the result of the best model architecture. Finally, to factorize the coverage of candidate selection and the accuracy of candidate ranking, we consider two evaluation scenarios: (1) the candidate list as derived from reaction center; (2) the above candidate list augmented with the true product if not found. This latter setup is marked with (*).

2.2.1 Results

Reaction Center Identification Table 2.1a reports the coverage of the model as compared to the real reaction core. Clearly, the coverage depends on the number of atom pairs K, with the higher coverage for larger values of K. These results demonstrate that even for $K = 8$, the model achieves high coverage, above 90%.

The results also clearly demonstrate the advantage of the global model over the local one, which is consistent across all experiments. The superiority of the global model is in line with the well-known fact that reactivity depends on more than the immediate local environment surrounding the reaction center. The presence of certain *functional groups* (structural motifs that appear frequently in organic chemistry)

USPTO-15K				
Method	$\lvert\theta\rvert$	**K=6**	**K=8**	**K=10**
Local	572K	80.1	85.0	87.7
Local	1003K	81.6	86.1	89.1
Global	756K	**86.7**	**90.1**	**92.2**
USPTO				
Local	572K	83.0	87.2	89.6
Global	756K	**89.8**	**92.0**	**93.3**
Avg. Num. of Candidates (USPTO)				
Template	-	482.3 out of 5006		
Global	-	60.9	246.5	1076

(a) Reaction Center Prediction Performance. Coverage is reported by picking the top K (K=6,8,10) reactivity pairs. $|\theta|$ is the number of model parameters.

USPTO-15K			
Method	**Cov.**	**P@1**	**P@5**
Coley et al.	100.0	72.1	90.7
WLN	90.1	74.9	86.3
WLDN	90.1	**76.7**	**86.8**
WLN (*)	100.0	81.4	94.8
WLDN (*)	100.0	**84.1**	**96.1**
USPTO			
Method	$\lvert\theta\rvert$	**P@1**	**P@5**
WLDN	3.2M	**79.6**	**89.2**
WLDN (*)	3.2M	83.9	95.2

(b) Candidate Ranking Performance. Precision at ranks 1,3,5 are reported. (*) denotes that the true product was added if not covered by the previous stage.

Table 2.1: Model Comparison on USPTO-15K and USPTO dataset.

far from the reaction center can promote or inhibit different modes of reactivity. Moreover, reactivity is often influenced by the presence of *reagents*, which are separate molecules that may not directly contribute atoms to the product. Consideration of both of these factors necessitates the use of a model that can account for long-range dependencies between atoms.

Figure 2-2 depicts one such example, where the observed reactivity can be attributed to the presence of a reagent molecule that is completely disconnected from the reaction center itself. While the local model fails to anticipate this reactivity, the global one accurately predicts the reaction center. The attention map highlights the reagent molecule as the determinant context.

Candidate Generation Here we compare the coverage of the generated candidates with the template-based model. Table 2.1a shows that for $K = 6$, our model generates an average of 60.1 candidates and reaches a coverage of 89.8%. The template-based baseline requires 5006 templates extracted from the training data (corresponding to a minimum of five precedent reactions) to achieve 90.1% coverage with an average of 482 candidates per example.

This weakness of the baseline model can be explained by the difficulty in defin-

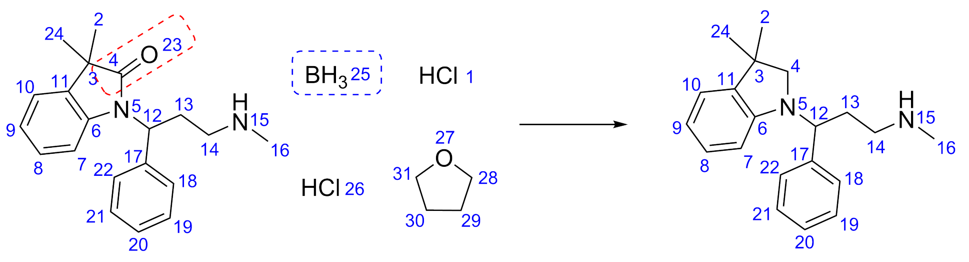
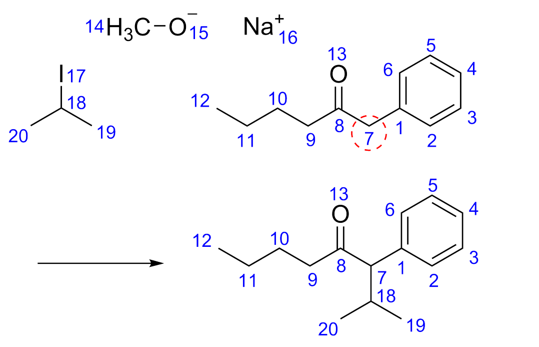


Figure 2-2: A reaction that reduces the carbonyl carbon of an amide by removing bond 4-23 (red circle). Reactivity at this site would be highly unlikely without the presence of borohydride (atom 25, blue circle). The global model correctly predicts bond 4-23 as the most susceptible to change, while the local model does not even include it in the top ten predictions. The attention map of the global model show that atoms 1, 25, and 26 were determinants of atom 4's predicted reactivity.

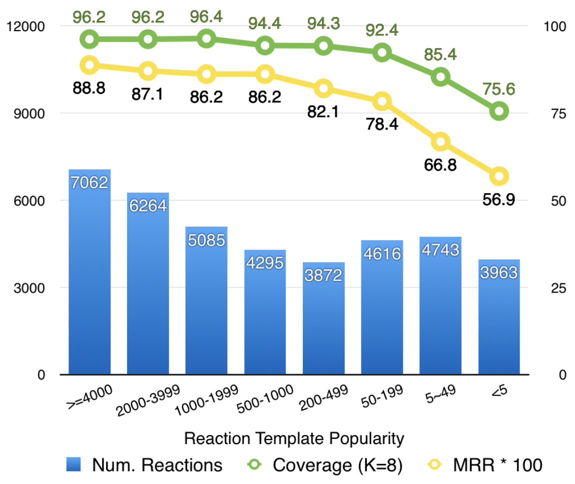
ing general heuristics with which to extract templates from reaction examples. It is possible to define different levels of specificity based on the extent to which atoms surrounding the reaction center are included or generalized [32]. This introduces an unavoidable trade-off between generality (fewer templates, higher coverage, more candidates) and specificity (more templates, less coverage, fewer candidates). Figure 2-3a illustrates one reaction example where the corresponding template is rare due to the adjacency of the reaction center to both a carbonyl group and a phenyl ring. Because adjacency to either group can influence reactivity, both are included as part of the template, although reactivity in this case does not require the additional specification of the phenyl group.

The massive number of templates required for high coverage is a serious impediment for the template approach because each template application requires solving a subgraph isomorphism problem. Specifically, it takes on average 7 seconds to apply the 5006 templates to a test instance, while our method takes less than 50 ms, about 140 times faster.

Candidate Ranking Table 2.1b reports the performance on the product prediction task. Since the baseline templates from [9] were optimized on the test and have 100% coverage, we compare its performance against our models to which the correct product is added (WLN(*) and WLDN(*)). Our model clearly outperforms the baseline by a wide margin. Even when compared against the candidates automatically computed from the reaction center, WLDN outperforms the baseline in top-1 accu-



(a) An example where reaction occurs at the α carbon (atom 7, red circle) of a carbonyl group (bond 8-13), also adjacent to a phenyl group (atoms 1-6). The corresponding template explicitly requires both the carbonyl and part of the phenyl ring as context (atoms 4, 7, 8, 13), although reactivity in this case does not require the additional specification of the phenyl group (atom 1).



(b) Performance of reactions with different popularity. MRR stands for mean reciprocal rank

Figure 2-3

racy. The results also demonstrate that the WLDN model consistently outperforms the WLN model. This is consistent with our intuition that modeling higher order dependencies between the difference vectors is advantageous over simply summing over them. Table 2.1b also shows the model performance improves when tested on the full USPTO dataset.

We further analyze model performance based on the frequency of the underlying transformation as reflected by the the number of template precedents. In Figure 2-3b we group the test instances according to their frequency and report the coverage of the global model and the mean reciprocal rank (MRR) of the WLDN model on each of them. As expected, our approach achieves the highest performance for frequent reactions. However, it maintains reasonable coverage and ranking accuracy even for rare reactions, which are particularly challenging for template-based methods.

2.2.2 Human Evaluation Study

We randomly selected 80 reaction examples from the test set, ten from each of the template popularity intervals of Figure 2-3b, and asked ten chemists to predict the outcome of each given its reactants. The average accuracy across the ten performers was 48.2%. Our model achieves an accuracy of 69.1%, very close to the best individual performer who scored 72.0%.

Chemist	56.3	50.0	72.0	63.8	66.3	65.0	40.0	58.8	25.0	16.3
Our Model	**69.1**									

Table 2.2: Human and model performance on 80 reactions randomly selected from the USPTO test set to cover a diverse range of reaction types. The first 8 are chemists with rich experience in organic chemistry (graduate, postdoctoral and professor level chemists) and the last two are graduate students in chemical engineering who use organic chemistry concepts regularly but have less formal training.

Chapter 3

Molecule Optimization

3.1 Background: Variational Autoencoder

The variational autoencoder [27] is a probabilistic generative model that learns both an encoder and a decoder for mapping data $\mathbf{x}$ to and from latent variables $\mathbf{z}$. The decoder defines the generative process of $\mathbf{x}$ given latent representation $\mathbf{z}$, i.e. the likelihood function $p_\theta(\mathbf{x}|\mathbf{z})$. The encoder approximates the posterior $p_\theta(\mathbf{z}|\mathbf{x}) \propto p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})$ with a parameterized model $q_\psi(\mathbf{z}|\mathbf{x})$. The encoder and decoder are jointly trained to maximize the evidence lower bound of the marginal likelihood

$$\mathcal{L}(\theta, \psi; \mathbf{x}) = \mathbb{E}_{q_\psi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\psi(\mathbf{z}|\mathbf{x})\right]$$

where prior distribution is a Gaussian $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$.

3.2 Junction Tree Variational Autoencoder

Our approach extends the variational autoencoder [27] to molecular graphs by introducing a suitable encoder and a matching decoder. Deviating from previous work [16, 30], we interpret each molecule as having been built from subgraphs chosen out of a vocabulary of valid components. These components are used as building blocks both when encoding a molecule into a vector representation as well as when decoding latent vectors back into valid molecular graphs. The key advantage of this view is that the decoder can realize a valid molecule piece by piece by utilizing the collection of valid components and how they interact, rather than trying to build the molecule atom by atom through chemically invalid intermediaries (Figure 1-3). An aromatic bond, for example, is chemically invalid on its own unless the entire aromatic ring is present. It would be therefore challenging to learn to build rings atom by atom rather than by introducing rings as part of the basic vocabulary.

Our vocabulary of components, such as rings, bonds and individual atoms, is chosen to be large enough so that a given molecule can be covered by overlapping components or *clusters* of atoms. The clusters serve the role analogous to cliques in graphical models, as they are expressive enough that a molecule can be covered by overlapping clusters without forming cluster cycles. In this sense, the clusters serve as cliques in a (non-optimal) triangulation of the molecular graph. We form a junction tree of such clusters and use it as the tree representation of the molecule. Since our choice of cliques is constrained a priori, we cannot guarantee that a junction tree exists with such clusters for an arbitrary molecule. However, our clusters are built on the basis of the molecules in the training set to ensure that a corresponding junction tree can be found. Empirically, our clusters cover most of the molecules in the test set.

The original molecular graph and its associated junction tree offer two complementary representations of a molecule. We therefore encode the molecule into a two-part latent representation $\mathbf{z} = [\mathbf{z}_\mathcal{T}, \mathbf{z}_G]$ where $\mathbf{z}_\mathcal{T}$ encodes the tree structure and what the clusters are in the tree without fully capturing how exactly the clusters are mutually connected. $\mathbf{z}_G$ encodes the graph to capture the fine-grained connectivity. Both parts are created by tree and graph encoders $q(\mathbf{z}_\mathcal{T}|\mathcal{T})$ and $q(\mathbf{z}_G|G)$. The latent representation is then decoded back into a molecular graph in two stages. As illustrated in Figure 3-1, we first reproduce the junction tree using a tree decoder $p(\mathcal{T}|\mathbf{z}_\mathcal{T})$ based on the information in $\mathbf{z}_\mathcal{T}$. Second, we predict the fine grain connectivity between the clusters in the junction tree using a graph decoder $p(G|\mathcal{T}, \mathbf{z}_G)$ to realize the full molecular graph. The junction tree approach allows us to maintain chemical feasibility during generation.

Notation A molecular graph is defined as $G = (V, E)$ where V is the set of atoms (vertices) and E the set of bonds (edges). Let $N(x)$ be the neighbor of x. We denote sigmoid function as $\sigma(\cdot)$ and ReLU function as $\tau(\cdot)$. We use i, j, k for nodes

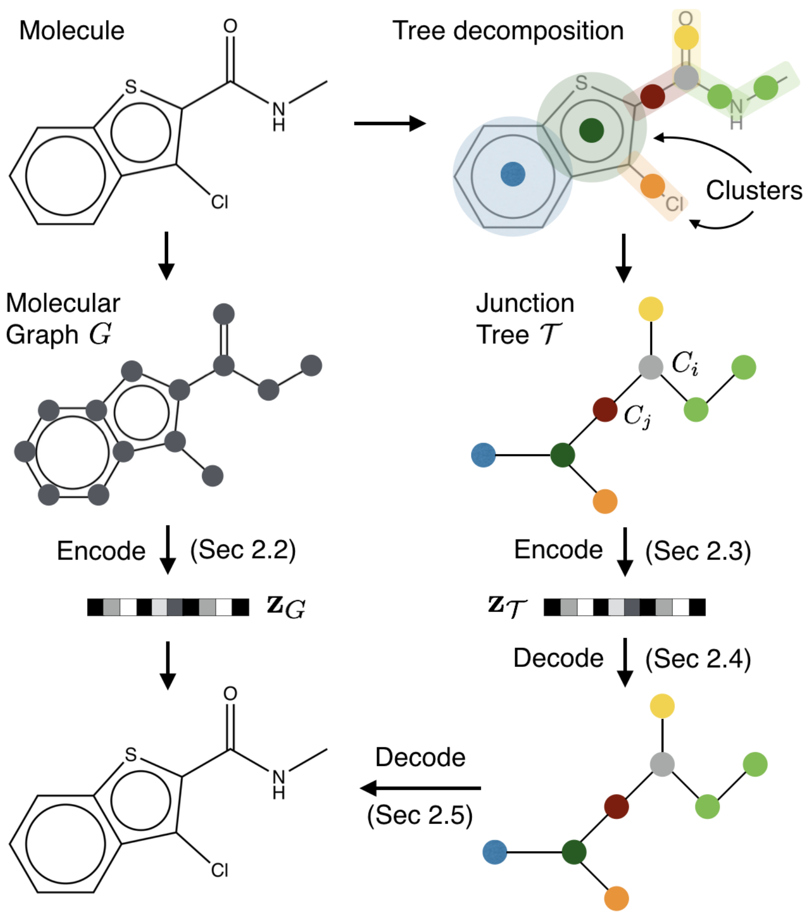


Figure 3-1: Overview of our molecule generation paradigm: A molecular graph G is first decomposed into its junction tree $\mathcal{T}_G$, where each colored node in the tree represents a substructure in the molecule. We then encode both the tree and graph into their latent embeddings $\mathbf{z}_\mathcal{T}$ and $\mathbf{z}_G$. To decode the molecule, we first reconstruct junction tree from $\mathbf{z}_\mathcal{T}$, and then assemble nodes in the tree back to original molecule, guided by $\mathbf{z}_G$.

in the tree and u, v, w for nodes in the graph.

3.2.1 Junction Tree

A tree decomposition maps a graph G into a *junction tree* by contracting certain vertices into a single node so that G becomes cycle-free. Formally, given a graph G, a junction tree $\mathcal{T}_G = (\mathcal{V}, \mathcal{E}, \mathcal{X})$ is a connected labeled tree whose node set is $\mathcal{V} = \{C_1, \cdots, C_n\}$ and edge set is $\mathcal{E}$. Each node or *cluster* $C_i = (V_i, E_i)$ is an induced subgraph of G, satisfying the following constraints:

1. The union of all clusters equals G. That is, $\bigcup_i V_i = V$ and $\bigcup_i E_i = E$.

2. Running intersection: For all clusters C_i, C_j and C_k, $V_i \cap V_j \subseteq V_k$ if C_k is on the path from C_i to C_j.

Viewing induced subgraphs as cluster labels, junction trees are labeled trees with label vocabulary $\mathcal{X}$. By our molecule tree decomposition, $\mathcal{X}$ contains only cycles (rings) and single edges. Thus the vocabulary size is limited ($|\mathcal{X}| = 780$ for a standard dataset with 250K molecules).

Tree Decomposition of Molecules Here we present our tree decomposition algorithm tailored for molecules, which finds its root in chemistry [41]. Our cluster vocabulary $\mathcal{X}$ includes chemical structures such as bonds and rings (Figure 3-1). Given a graph G, we first find all its simple cycles, and its edges not belonging to any cycles. Two simple rings are merged together if they have more than two overlapping atoms, as they constitute a specific structure called bridged compounds [8]. Each of those cycles or edges is considered as a cluster. Next, a cluster graph is constructed by adding edges between all intersecting clusters. Finally, we select one of its spanning trees as the junction tree of G (Figure 3-1). As a result of ring merging, any two clusters in the junction tree have at most two atoms in common, facilitating efficient inference in the graph decoding phase. The detailed procedure is described in the supplementary.

3.2.2 Graph Encoder

We first encode the latent representation of G by a graph message passing network [10, 15]. Each vertex v has a feature vector $\mathbf{x}_v$ indicating the atom type, valence, and other properties. Similarly, each edge $(u, v) \in E$ has a feature vector $\mathbf{x}_{uv}$ indicating its bond type, and two hidden vectors $\boldsymbol{\nu}_{uv}$ and $\boldsymbol{\nu}_{vu}$ denoting the message from u to v and vice versa. Due to the loopy structure of the graph, messages are exchanged in a loopy belief propagation fashion:

$$\boldsymbol{\nu}_{uv}^{(t)} = \tau(\mathbf{W}_1^g \mathbf{x}_u + \mathbf{W}_2^g \mathbf{x}_{uv} + \mathbf{W}_3^g \sum_{w \in N(u) \setminus v} \boldsymbol{\nu}_{wu}^{(t-1)}) \tag{3.1}$$

where $\boldsymbol{\nu}_{uv}^{(t)}$ is the message computed in t-th iteration, initialized with $\boldsymbol{\nu}_{uv}^{(0)} = \mathbf{0}$. After T steps of iteration, we aggregate those messages as the latent vector of each vertex, which captures its local graphical structure:

$$\mathbf{h}_u = \tau(\mathbf{U}_1^g \mathbf{x}_u + \sum\nolimits_{v \in N(u)} \mathbf{U}_2^g \boldsymbol{\nu}_{vu}^{(T)}) \tag{3.2}$$

The final graph representation is $\mathbf{h}_G = \sum_i \mathbf{h}_i / |V|$. The mean $\boldsymbol{\mu}_G$ and log variance $\log \boldsymbol{\sigma}_G$ of the variational posterior approximation are computed from $\mathbf{h}_G$ with two separate affine layers. $\mathbf{z}_G$ is sampled from a Gaussian $\mathcal{N}(\boldsymbol{\mu}_G, \boldsymbol{\sigma}_G)$.

3.2.3 Tree Encoder

We similarly encode $\mathcal{T}_G$ with a tree message passing network. Each cluster C_i is represented by a one-hot encoding $\mathbf{x}_i$ representing its label type. Each edge (C_i, C_j) is associated with two message vectors $\mathbf{m}_{ij}$ and $\mathbf{m}_{ji}$. We pick an arbitrary leaf node as the root and propagate messages in two phases. In the first bottom-up phase, messages are initiated from the leaf nodes and propagated iteratively towards root. In the top-down phase, messages are propagated from the root to all the leaf nodes. Message $\mathbf{m}_{ij}$ is updated as:

$$\mathbf{m}_{ij} = \mathrm{GRU}(\mathbf{x}_i, \{\mathbf{m}_{ki}\}_{k \in N(i) \backslash j}) \tag{3.3}$$

where GRU is a Gated Recurrent Unit [7, 34] adapted for tree message passing:

$$\begin{aligned}
\mathbf{s}_{ij} &= \sum\nolimits_{k \in N(i) \backslash j} \mathbf{m}_{ki} && (3.4)\\
\mathbf{z}_{ij} &= \sigma(\mathbf{W}^z \mathbf{x}_i + \mathbf{U}^z \mathbf{s}_{ij} + \mathbf{b}^z) && (3.5)\\
\mathbf{r}_{ki} &= \sigma(\mathbf{W}^r \mathbf{x}_i + \mathbf{U}^r \mathbf{m}_{ki} + \mathbf{b}^r) && (3.6)\\
\widetilde{\mathbf{m}}_{ij} &= \tanh(\mathbf{W} \mathbf{x}_i + \mathbf{U} \sum_{k \in N(i) \backslash j} \mathbf{r}_{ki} \odot \mathbf{m}_{ki}) && (3.7)\\
\mathbf{m}_{ij} &= (1 - \mathbf{z}_{ij}) \odot \mathbf{s}_{ij} + \mathbf{z}_{ij} \odot \widetilde{\mathbf{m}}_{ij} && (3.8)
\end{aligned}$$

The message passing follows the schedule where $\mathbf{m}_{ij}$ is computed only when all its precursors $\{\mathbf{m}_{ki} \mid k \in N(i) \backslash j\}$ have been computed. This architectural design is motivated by the belief propagation algorithm over trees and is thus different from the graph encoder.

After the message passing, we obtain the latent representation of each node $\mathbf{h}_i$ by aggregating its inward messages:

$$\mathbf{h}_i = \tau(\mathbf{W}^o \mathbf{x}_i + \sum\nolimits_{k \in N(i)} \mathbf{U}^o \mathbf{m}_{ki}) \tag{3.9}$$

The final tree representation is $\mathbf{h}_{\mathcal{T}_G} = \mathbf{h}_{root}$, which encodes a rooted tree $(\mathcal{T}, root)$. Unlike the graph encoder, we do not apply node average pooling because it confuses the tree decoder which node to generate first. $\mathbf{z}_{\mathcal{T}_G}$ is sampled in a similar way as in the graph encoder. For simplicity, we abbreviate $\mathbf{z}_{\mathcal{T}_G}$ as $\mathbf{z}_{\mathcal{T}}$ from now on.

This tree encoder plays *two* roles in our framework. First, it is used to compute $\mathbf{z}_{\mathcal{T}}$, which only requires the bottom-up phase of the network. Second, after a tree $\widehat{\mathcal{T}}$ is decoded from $\mathbf{z}_{\mathcal{T}}$, it is used to compute messages $\widehat{\mathbf{m}}_{ij}$ over the entire $\widehat{\mathcal{T}}$, to provide essential contexts of every node during graph decoding. This requires both top-down and bottom-up phases. We will elaborate this in section 3.2.5.

3.2.4 Tree Decoder

We decode a junction tree $\mathcal{T}$ from its encoding $\mathbf{z}_{\mathcal{T}}$ with a tree structured decoder. The tree is constructed in a top-down fashion by generating one node at a time. As illustrated in Figure 3-2, our tree decoder traverses the entire tree from the root, and generates nodes in their depth-first order. For every visited node, the decoder first makes a *topological prediction*: whether this node has children to be generated. When a new child node is created, we predict its label and recurse this process. Recall that cluster labels represent subgraphs in a molecule. The decoder backtracks when a node has no more children to generate.

At each time step, a node receives information from other nodes in the current tree for making those predictions. The information is propagated through message vectors $\mathbf{h}_{ij}$ when trees are incrementally constructed. Formally, let $\tilde{\mathcal{E}} = \{(i_1, j_1), \cdots, (i_m, j_m)\}$ be the edges traversed in a depth first traversal over $\mathcal{T} = (\mathcal{V}, \mathcal{E})$, where $m = 2|\mathcal{E}|$ as each edge is traversed in both directions. The model visits node i_t at time t. Let $\tilde{\mathcal{E}}_t$ be the first t edges in $\tilde{\mathcal{E}}$. The message $\mathbf{h}_{i_t, j_t}$ is updated through previous messages:

$$\mathbf{h}_{i_t, j_t} = \mathrm{GRU}(\mathbf{x}_{i_t}, \{\mathbf{h}_{k, i_t}\}_{(k, i_t) \in \tilde{\mathcal{E}}_t, k \neq j_t}) \tag{3.10}$$

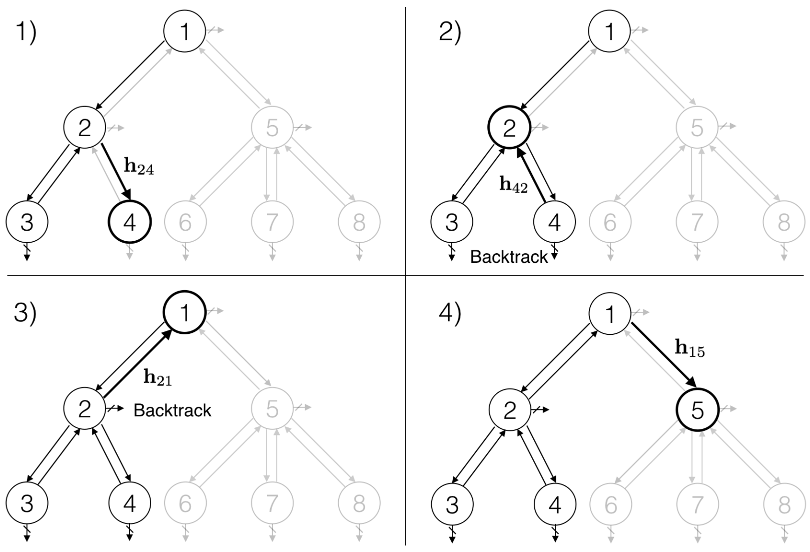


Figure 3-2: Illustration of the tree decoding process. Nodes are labeled in the order in which they are generated. 1) Node 2 expands child node 4 and predicts its label with message $\mathbf{h}_{24}$. 2) As node 4 is a leaf node, decoder backtracks and computes message $\mathbf{h}_{42}$. 3) Decoder continues to backtrack as node 2 has no more children. 4) Node 1 expands node 5 and predicts its label.

where GRU is the same recurrent unit as in the tree encoder.

Topological Prediction When the model visits node i_t, it makes a binary prediction on whether it still has children to be generated. We compute this probability by combining $\mathbf{z}_\mathcal{T}$, node features $\mathbf{x}_{i_t}$ and inward messages $\mathbf{h}_{k,i_t}$ via a one hidden layer network followed by a sigmoid function:

$$p_t = \sigma(\mathbf{u}^d \cdot \tau(\mathbf{W}_1^d \mathbf{x}_{i_t} + \mathbf{W}_2^d \mathbf{z}_\mathcal{T} + \mathbf{W}_3^d \sum_{(k,i_t)\in\tilde{\mathcal{E}}_t} \mathbf{h}_{k,i_t}) \tag{3.11}$$

Label Prediction When a child node j is generated from its parent i, we predict its node label with

$$\mathbf{q}_j = \mathrm{softmax}(\mathbf{U}^l \tau(\mathbf{W}_1^l \mathbf{z}_\mathcal{T} + \mathbf{W}_2^l \mathbf{h}_{ij})) \tag{3.12}$$

where $\mathbf{q}_j$ is a distribution over label vocabulary $\mathcal{X}$. When j is a root node, its parent i is a virtual node and $\mathbf{h}_{ij} = \mathbf{0}$.

Learning The tree decoder aims to maximize the likelihood $p(\mathcal{T}|\mathbf{z}_\mathcal{T})$. Let $\hat{p}_t \in \{0,1\}$ and $\hat{\mathbf{q}}_j$ be the ground truth topological and label values, the decoder minimizes

Algorithm 1 Tree decoding at sampling time

```
Require: Latent representation $\mathbf{z}_\mathcal{T}$
 1: Initialize: Tree $\widehat{\mathcal{T}} \leftarrow \emptyset$
 2: function SampleTree($i, t$)
 3:     Set $\mathcal{X}_i \leftarrow$ all cluster labels that are chemically compatible with node $i$ and its
        current neighbors.
 4:     Set $d_t \leftarrow$ expand with probability $p_t$.                    ▷ Eq.(3.11)
 5:     if $d_t = expand$ and $\mathcal{X}_i \neq \emptyset$ then
 6:         Create a node $j$ and add it to tree $\widehat{\mathcal{T}}$.
 7:         Sample the label of node $j$ from $\mathcal{X}_i$                  ▷. Eq.(3.12)
 8:         SampleTree($j, t+1$)
 9:     end if
10: end function
```

the following cross entropy loss:[1]

$$\mathcal{L}_c(\mathcal{T}) = \sum\nolimits_t \mathcal{L}^d(p_t, \hat{p}_t) + \sum\nolimits_j \mathcal{L}^l(\mathbf{q}_j, \hat{\mathbf{q}}_j) \tag{3.13}$$

Similar to sequence generation, during training we perform *teacher forcing*: after topological and label prediction at each step, we replace them with their ground truth so that the model makes predictions given correct histories.

Decoding & Feasibility Check Algorithm 1 shows how a tree is sampled from $\mathbf{z}_\mathcal{T}$. The tree is constructed recursively guided by topological predictions without any external guidance used in training. To ensure the sampled tree could be realized into a valid molecule, we define set $\mathcal{X}_i$ to be cluster labels that are chemically compatible with node i and its current neighbors. When a child node j is generated from node i, we sample its label from $\mathcal{X}_i$ with a renormalized distribution $\mathbf{q}_j$ over $\mathcal{X}_i$ by masking out invalid labels.

3.2.5 Graph Decoder

The final step of our model is to reproduce a molecular graph G that underlies the predicted junction tree $\widehat{\mathcal{T}} = (\widehat{\mathcal{V}}, \widehat{\mathcal{E}})$. Note that this step is not deterministic since there are potentially many molecules that correspond to the same junction tree. The underlying degree of freedom pertains to how neighboring clusters C_i and C_j are

[1]The node ordering is not unique as the order within sibling nodes is ambiguous. In this paper we train our model with one ordering and leave this issue for future work.

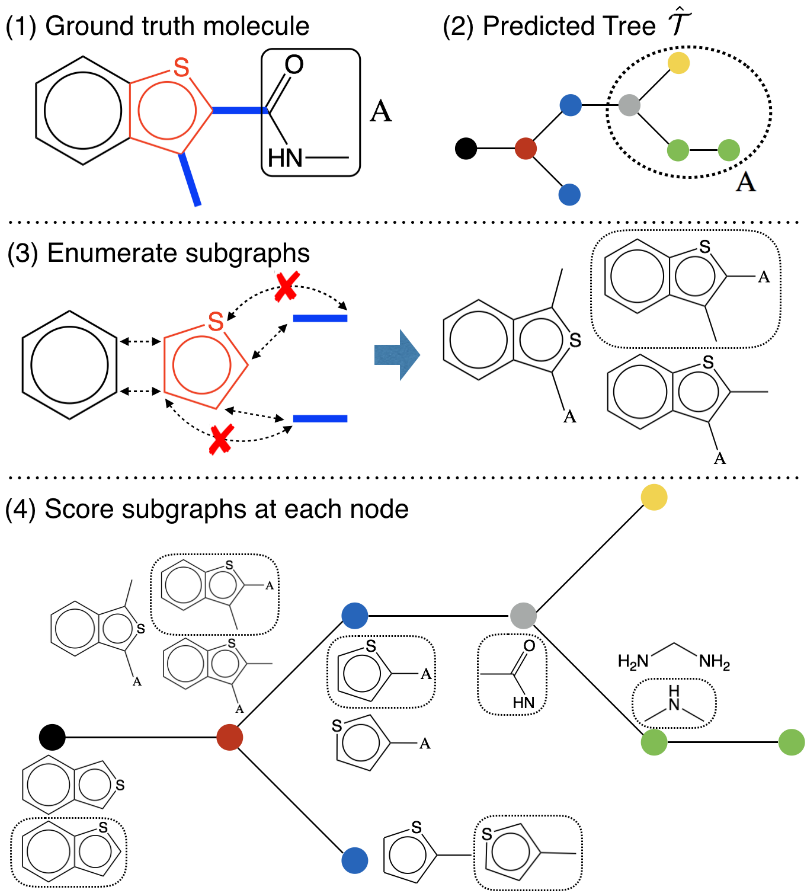

(1) Ground truth molecule
(2) Predicted Tree $\hat{\mathcal{T}}$
A
A
(3) Enumerate subgraphs
(4) Score subgraphs at each node

Figure 3-3: Decode a molecule from a junction tree. 1) Ground truth molecule G. 2) Predicted junction tree $\widehat{\mathcal{T}}$. 3) We enumerate different combinations between red cluster C and its neighbors. Crossed arrows indicate combinations that lead to chemically infeasible molecules. Note that if we discard tree structure during enumeration (i.e., ignoring subtree A), the last two candidates will collapse into the same molecule. 4) Rank subgraphs at each node. The final graph is decoded by putting together all the predicted subgraphs (dashed box).

attached to each other as subgraphs. Our goal here is to assemble the subgraphs (nodes in the tree) together into the correct molecular graph.

Let $\mathcal{G}(\mathcal{T})$ be the set of graphs whose junction tree is $\mathcal{T}$. Decoding graph $\hat{G}$ from $\widehat{\mathcal{T}} = (\widehat{\mathcal{V}}, \widehat{\mathcal{E}})$ is a structured prediction:

$$\hat{G} = \arg \max_{G' \in \mathcal{G}(\widehat{\mathcal{T}})} f^a(G') \tag{3.14}$$

where f^a is a scoring function over candidate graphs. We only consider scoring functions that decompose across the clusters and their neighbors. In other words, each term in the scoring function depends only on how a cluster C_i is attached to its neighboring clusters C_j, $j \in N_{\widehat{\mathcal{T}}}(i)$ in the tree $\widehat{\mathcal{T}}$. The problem of finding the highest scoring graph $\hat{G}$ – the assembly task – could be cast as a graphical model inference task in a model induced by the junction tree. However, for efficiency reasons, we will assemble the molecular graph one neighborhood at a time, following the order in which the tree itself was decoded. In other words, we start by sampling the assembly of the root and its neighbors according to their scores. Then we proceed to assemble the neighbors and their associated clusters (removing the degrees of freedom set by the root assembly), and so on.

It remains to be specified how each neighborhood realization is scored. Let G_i be the subgraph resulting from a particular merging of cluster C_i in the tree with its neighbors C_j, $j \in N_{\widehat{\mathcal{T}}}(i)$. We score G_i as a candidate subgraph by first deriving a vector representation $\mathbf{h}_{G_i}$ and then using $f_i^a(G_i) = \mathbf{h}_{G_i} \cdot \mathbf{z}_G$ as the subgraph score. To this end, let u, v specify atoms in the candidate subgraph G_i and let $\alpha_v = i$ if $v \in C_i$ and $\alpha_v = j$ if $v \in C_j \backslash C_i$. The indices α_v are used to mark the position of the atoms in the junction tree, and to retrieve messages $\widehat{\mathbf{m}}_{i,j}$ summarizing the subtree under i along the edge (i, j) obtained by running the tree encoding algorithm. The neural messages pertaining to the atoms and bonds in subgraph G_i are obtained and aggregated into $\mathbf{h}_{G_i}$, similarly to the encoding step, but with different (learned) parameters:

$$\begin{aligned}
\boldsymbol{\mu}_{uv}^{(t)} &= \tau(\mathbf{W}_1^a \mathbf{x}_u + \mathbf{W}_2^a \mathbf{x}_{uv} + \mathbf{W}_3^a \widetilde{\boldsymbol{\mu}}_{uv}^{(t-1)}) \\
\widetilde{\boldsymbol{\mu}}_{uv}^{(t-1)} &= \begin{cases} \sum_{w \in N(u)\backslash v} \boldsymbol{\mu}_{wu}^{(t-1)} & \alpha_u = \alpha_v \\ \widehat{\mathbf{m}}_{\alpha_u,\alpha_v} + \sum_{w \in N(u)\backslash v} \boldsymbol{\mu}_{wu}^{(t-1)} & \alpha_u \neq \alpha_v \end{cases}
\end{aligned} \tag{3.15}$$

The major difference from Eq. (3.1) is that we augment the model with tree messages $\widehat{\mathbf{m}}_{\alpha_u,\alpha_v}$ derived by running the tree encoder over the predicted tree $\widehat{\mathcal{T}}$. $\widehat{\mathbf{m}}_{\alpha_u,\alpha_v}$ provides a tree dependent positional context for bond (u, v) (illustrated as subtree A in Figure 3-3).

Learning The graph decoder parameters are learned to maximize the log-likelihood of predicting correct subgraphs G_i of the ground true graph G at each

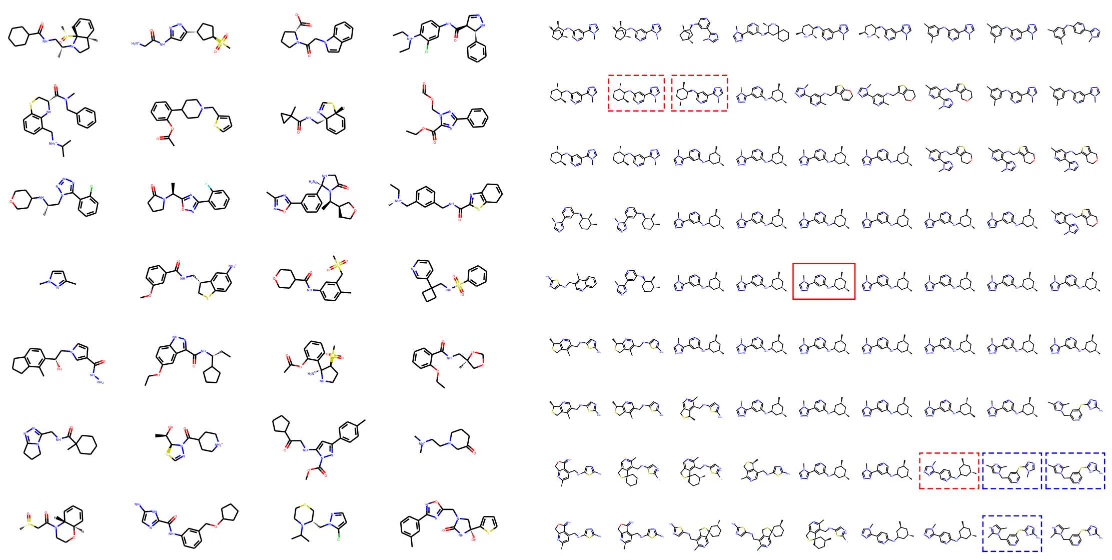


Figure 3-4: **Left**: Random molecules sampled from prior distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$. **Right**: Visualization of the local neighborhood of a molecule in the center. Three molecules highlighted in red dashed box have the same tree structure as the center molecule, but with different graph structure as their clusters are combined differently. The same phenomenon emerges in another group of molecules (blue dashed box).

tree node:

$$\mathcal{L}_g(G) = \sum_i \left[f^a(G_i) - \log \sum_{G'_i \in \mathcal{G}_i} \exp(f^a(G'_i)) \right] \tag{3.16}$$

where $\mathcal{G}_i$ is the set of possible candidate subgraphs at tree node i. During training, we again apply teacher forcing, i.e. we feed the graph decoder with ground truth trees as input.

Complexity By our tree decomposition, any two clusters share at most two atoms, so we only need to merge at most two atoms or one bond. By pruning chemically invalid subgraphs and merging isomorphic graphs, $|\mathcal{G}_i| \approx 4$ on average when tested on a standard ZINC drug dataset. The computational complexity of JT-VAE is therefore linear in the number of clusters, scaling nicely to large graphs.

3.3 Experiments

Our evaluation efforts measure various aspects of molecular generation. The first two evaluations follow previously proposed tasks [30]. We also introduce a third task —

constrained molecule optimization.

- **Molecule reconstruction and validity** We test the VAE models on the task of reconstructing input molecules from their latent representations, and decoding valid molecules when sampling from prior distribution. (Section 3.3.1)

- **Bayesian optimization** Moving beyond generating valid molecules, we test how the model can produce novel molecules with desired properties. To this end, we perform Bayesian optimization in the latent space to search molecules with specified properties. (Section 3.3.2)

- **Constrained molecule optimization** The task is to modify given molecules to improve specified properties, while constraining the degree of deviation from the original molecule. This is a more realistic scenario in drug discovery, where development of new drugs usually starts with known molecules such as existing drugs [3]. Since it is a new task, we cannot compare to any existing baselines. (Section 3.3.3)

Below we describe the data, baselines and model configuration that are shared across the tasks. Additional setup details are provided in the task-specific sections.

Data We use the ZINC molecule dataset from Kusner et al. [30] for our experiments. It contains about 250K drug molecules extracted from the ZINC database [49]. We follow the same training/testing split as the previous work.

Baselines We compare our approach with SMILES-based baselines: 1) Character VAE (CVAE) [16] which generates SMILES strings character by character; 2) Grammar VAE (GVAE) [30] that generates SMILES following syntactic constraints given by a context-free grammar; 3) Syntax-directed VAE (SD-VAE) [11] that incorporates both syntactic and semantic constraints of SMILES via attribute grammar. For molecule generation task, we also compare with GraphVAE [47] that directly generates atom labels and adjacency matrices of graphs.

Model Configuration To be comparable with the above baselines, we set the latent space dimension as 56, i.e., the tree and graph representation $\mathbf{h}_{\mathcal{T}}$ and $\mathbf{h}_G$ have 28 dimensions each. Full training details and model configurations are provided in the appendix.

Method	**Reconstruction**	**Validity**
CVAE	44.6%	0.7%
GVAE	53.7%	7.2%
SD-VAE[2]	76.2%	43.5%
GraphVAE	-	13.5%
JT-VAE	**76.7%**	**100.0%**

Table 3.1: Reconstruction accuracy and prior validity results. Baseline results are copied from Kusner et al. [30], Dai et al. [11], Simonovsky and Komodakis [47].

3.3.1 Molecule Reconstruction and Validity

Setup The first task is to reconstruct and sample molecules from latent space. Since both encoding and decoding process are stochastic, we estimate reconstruction accuracy by Monte Carlo method used in [30]: Each molecule is encoded 10 times and each encoding is decoded 10 times. We report the portion of the 100 decoded molecules that are identical to the input molecule. For a fair comparison, we define two molecules as identical if they have the same SMILES string. At testing time, we convert all generated graphs to SMILES using RDKit.

To compute validity, we sample 1000 latent vectors from the prior distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$, and decode each of these vectors 100 times. We report the percentage of decoded molecules that are chemically valid (checked by RDKit).

Results Table 3.1 shows that JT-VAE outperforms previous models in molecule reconstruction, and **always** produces valid molecules when sampled from prior distribution. These sampled molecules have non-trivial structures such as simple chains (Figure 3-4). We further sampled 5000 molecules from prior and found they are *all distinct* from the training set. Thus our model is not a simple memorization.

Analysis We qualitatively examine the latent space of JT-VAE by visualizing the neighborhood of molecules. Given a molecule, we follow the method in Kusner et al. [30] to construct a grid visualization of its neighborhood. For comparison, we select the same molecule visualized in Dai et al. [11]. Figure 3-4 shows the local neighborhood of this molecule. Compared to the figure in Dai et al. [11], our neighborhood does not contain molecules with huge rings (with more than 7 atoms), which rarely occur in the dataset. We also highlight two groups of closely resembling molecules

[2]The SD-VAE result is copied from Table 1 in Dai et al. [11].

Method	**1st**	**2nd**	**3rd**
CVAE	1.98	1.42	1.19
GVAE	2.94	2.89	2.80
SD-VAE	4.04	3.50	2.96
JT-VAE	**5.30**	**4.93**	**4.49**

Table 3.2: Best molecule property scores found by each method. Baseline results are from Kusner et al. [30], Dai et al. [11].

that have identical tree structures but vary only in how clusters are attached together. This demonstrates the smoothness of learned molecular embeddings.

3.3.2 Bayesian Optimization

Setup The second task is to produce novel molecules with desired properties. Following [30], our target chemical property $y(\cdot)$ is octanol-water partition coefficients (logP) penalized by the synthetic accessibility (SA) score and number of long cycles.[3] To perform Bayesian optimization (BO), we first train a VAE and associate each molecule with a latent vector, given by the mean of the variational encoding distribution. After the VAE is learned, we train a sparse Gaussian process (SGP) to predict $y(m)$ given its latent representation. Then we perform five iterations of batched BO using the expected improvement heuristic.

For comparison, we report 1) the predictive performance of SGP trained on latent encodings learned by different VAEs, measured by log-likelihood (LL) and root mean square error (RMSE) with 10-fold cross validation. 2) The top-3 molecules found by BO under different models.

Results As shown in Table 3.2, JT-VAE finds molecules with significantly better scores than previous methods. Figure 3-5 lists the top-3 best molecules found by JT-VAE. In fact, JT-VAE finds over 50 molecules with scores over 3.50 (the second best molecule proposed by SD-VAE). Moreover, the SGP yields better predictive performance when trained on JT-VAE embeddings (Table 3.3).

[3] $y(m) = logP(m) - SA(m) - cycle(m)$ where $cycle(m)$ counts the number of rings that have more than six atoms.

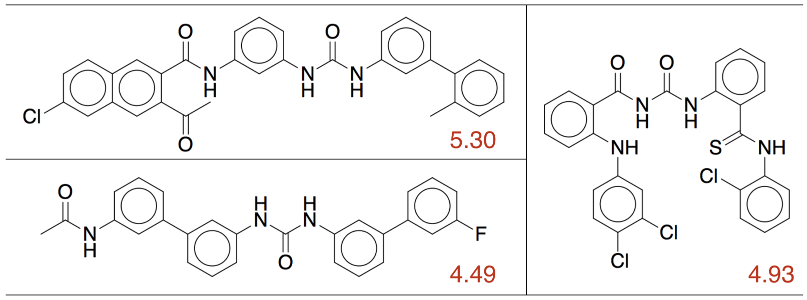


Figure 3-5: Best three molecules and their property scores found by JT-VAE using Bayesian optimization.

Method	LL	RMSE
CVAE	-1.812 ± 0.004	1.504 ± 0.006
GVAE	-1.739 ± 0.004	1.404 ± 0.006
SD-VAE	-1.697 ± 0.015	1.366 ± 0.023
JT-VAE	$\mathbf{-1.658 \pm 0.023}$	$\mathbf{1.290 \pm 0.026}$

Table 3.3: Predictive performance of sparse Gaussian Processes trained on different VAEs. Baseline results are copied from Kusner et al. [30] and Dai et al. [11].

3.3.3 Constrained Optimization

Setup The third task is to perform molecule optimization in a constrained scenario. Given a molecule m, the task is to find a different molecule m' that has the highest property value with the molecular similarity $sim(m, m') \geq \delta$ for some threshold δ. We use Tanimoto similarity with Morgan fingerprint [42] as the similarity metric, and penalized logP coefficient as our target chemical property. For this task, we jointly train a property predictor F (parameterized by a feed-forward network) with JT-VAE to predict $y(m)$ from the latent embedding of m. To optimize a molecule m, we start from its latent representation, and apply gradient ascent in the latent space to improve the predicted score $F(\cdot)$, similar to [38]. After applying $K = 80$ gradient steps, K molecules are decoded from resulting latent trajectories, and we report the molecule with the highest $F(\cdot)$ that satisfies the similarity constraint. A modification succeeds if one of the decoded molecules satisfies the constraint and is distinct from the original.

To provide the greatest challenge, we selected 800 molecules with the *lowest* property score $y(\cdot)$ from the test set. We report the success rate (how often a modification

δ	**Improvement**	**Similarity**	**Success**
0.0	1.91 ± 2.04	0.28 ± 0.15	97.5%
0.2	1.68 ± 1.85	0.33 ± 0.13	97.1%
0.4	0.84 ± 1.45	0.51 ± 0.10	83.6%
0.6	0.21 ± 0.71	0.69 ± 0.06	46.4%

Table 3.4: Constrained optimization result of JT-VAE: mean and standard deviation of property improvement, molecular similarity and success rate under constraints $sim(m, m') \geq \delta$ with varied δ.

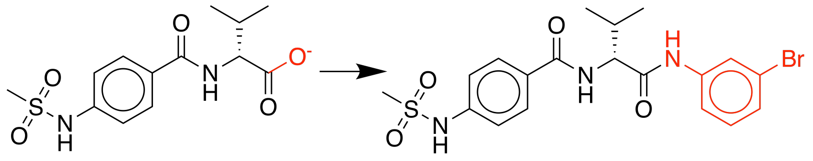


Figure 3-6: A molecule modification that yields an improvement of 4.0 with molecular similarity 0.617 (modified part is in red).

succeeds), and among success cases the average improvement $y(m')-y(m)$ and molecular similarity $sim(m, m')$ between the original and modified molecules m and m'.

Results Our results are summarized in Table 3.4. The unconstrained scenario ($\delta = 0$) has the best average improvement, but often proposes dissimilar molecules. When we tighten the constraint to $\delta = 0.4$, about 80% of the time our model finds similar molecules, with an average improvement 0.84. This also demonstrates the smoothness of the learned latent space. Figure 3-6 illustrates an effective modification resulting in a similar molecule with great improvement.

Chapter 4

Related Work

4.1 Graph Neural Network

Graph-structured Encoders The neural network formulation on graphs was first proposed by Gori et al. [17], Scarselli et al. [43], and later enhanced by Li et al. [34] with gated recurrent units. For recurrent architectures over graphs, **?**] designed Weisfeiler-Lehman kernel network inspired by graph kernels. Dai et al. [10] considered a different architecture where graphs were viewed as latent variable graphical models, and derived their model from message passing algorithms. Our tree and graph encoder are closely related to this graphical model perspective, and to neural message passing networks [15]. For convolutional architectures, Duvenaud et al. [13] introduced a convolution-like propagation on molecular graphs, which was generalized to other domains by Niepert et al. [39]. Bruna et al. [4], Henaff et al. [20] developed graph convolution in spectral domain via graph Laplacian. For applications, graph neural networks are used in semi-supervised classification [29], computer vision [37], and chemical domains [25, 44, 22].

4.2 Reaction Prediction

Template-based Approach Existing machine learning models for product prediction are mostly built on reaction templates. These approaches differ in the way templates are specified and in the way the final product is selected from multiple candidates. For instance, Wei et al. [55] learns to select among 16 pre-specified, hand-encoded templates, given fingerprints of reactants and reagents. While this work was developed on a narrow range of chemical reaction types, it is among the first implementations that demonstrates the potential of neural models for analyzing chemical reactions.

More recent work has demonstrated the power of neural methods on a broader set of reactions. For instance, Segler and Waller [45] and Coley et al. [9] use a data-driven approach to obtain a large set of templates, and then employ a neural model to rank the candidates. The key difference between these approaches is the representation of the reaction. In Segler and Waller [45], molecules are represented based on their Morgan fingerprints, while Coley et al. [9] represents reactions by the features of atoms and bonds in the reaction center. However, the template-based architecture limits both of these methods in scaling up to larger datasets with more diversity.

Template-free Approach Kayala et al. [24] also presented a template-free approach to predict reaction outcomes. Our approach differs from theirs in several ways. First, Kayala et al. operates at the mechanistic level - identifying elementary mechanistic steps rather than the overall transformations from reactants to products. Since most reactions consist of many mechanistic steps, their approach requires multiple predictions to fulfill an entire reaction. Our approach operates at the graph level - predicting transformations from reactants to products in a single step. Second, mechanistic descriptions of reactions are not given in existing reaction databases. Therefore, Kayala et al. created their training set based on a mechanistic-level template-driven expert system. In contrast, our model is learned directly from real-world experimental data. Third, Kayala et al. uses feed-forward neural networks where atoms and graphs are represented by molecular fingerprints and additional hand-crafted features. Our approach builds from graph neural networks to encode graph structures.

4.3 Molecule Optimization

Molecule Generation Previous work on molecule generation mostly operates on SMILES strings. Gómez-Bombarelli et al. [16], Segler et al. [46] built generative models of SMILES strings with recurrent decoders. Unfortunately, these models could generate invalid SMILES that do not result in any molecules. To remedy this issue, Kusner et al. [30], Dai et al. [11] complemented the decoder with syntactic and semantic constraints of SMILES by context free and attribute grammars, but these grammars do not fully capture chemical validity. Other techniques such as active learning [21] and reinforcement learning [18] encourage the model to generate valid SMILES through additional training signal. Very recently, Simonovsky and Komodakis [47] proposed to generate molecular graphs by predicting their adjacency matrices, and Li et al. [35] generated molecules node by node. In comparison, our method enforces chemical validity and is more efficient due to the coarse-to-fine generation.

Tree-structured Models Our tree encoder is related to recursive neural networks and tree-LSTM [48, 51, 57]. These models encode tree structures where nodes in the tree are bottom-up transformed into vector representations. In contrast, our model propagates information both bottom-up and top-down.

On the decoding side, tree generation naturally arises in natural language parsing [14, 28]. Different from our approach, natural language parsers have access to input words and only predict the topology of the tree. For general purpose tree generation, Vinyals et al. [53], Aharoni and Goldberg [1] applied recurrent networks to generate linearized version of trees, but their architectures were entirely sequence-based. Dong and Lapata [12], Alvarez-Melis and Jaakkola [2] proposed tree-based architectures that construct trees top-down from the root. Our model is most closely related to Alvarez-Melis and Jaakkola [2] that disentangles topological prediction from label prediction, but we generate nodes in a depth-first order and have additional steps that propagate information bottom-up. This forward-backward propagation also appears in Parisotto et al. [40], but their model is node based whereas ours is based on message passing.

Chapter 5

Conclusion

In this thesis, we proposed a novel template-free approach for chemical reaction prediction. Instead of generating candidate products by reaction templates, we first predict a small set of atoms/bonds in reaction center, and then produce candidate products by enumerating all possible bond configuration changes within the set. Compared to template based approach, our framework runs 140 times faster, allowing us to scale to much larger reaction databases. Both our reaction center identifier and candidate ranking model build from Weisfeiler-Lehman Network and its variants that learn compact representation of graphs and reactions.

Furthermore, we presented a junction tree variational autoencoder for generating molecular graphs. Our method significantly outperforms previous work in molecule generation and optimization. For future work, we attempt to generalize our method for general low-treewidth graphs.

Bibliography

[1] Roee Aharoni and Yoav Goldberg. Towards string-to-tree neural machine translation. *arXiv preprint arXiv:1704.04743*, 2017.

[2] David Alvarez-Melis and Tommi S Jaakkola. Tree-structured decoding with doubly-recurrent neural networks. 2016.

[3] Jérémy Besnard, Gian Filippo Ruda, Vincent Setola, Keren Abecassis, Ramona M Rodriguiz, Xi-Ping Huang, Suzanne Norval, Maria F Sassano, Antony I Shin, Lauren A Webster, et al. Automated design of ligands to polypharmacological profiles. *Nature*, 492(7428):215–220, 2012.

[4] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[5] Jonathan H Chen and Pierre Baldi. No electron left behind: a rule-based expert system to predict chemical reactions and reaction mechanisms. *Journal of chemical information and modeling*, 49(9):2034–2043, 2009.

[6] Clara D Christ, Matthias Zentgraf, and Jan M Kriegl. Mining electronic laboratory notebooks: analysis, retrosynthesis, and reaction based enumeration. *Journal of chemical information and modeling*, 52(7):1745–1756, 2012.

[7] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[8] Jonathan Clayden, Nick Greeves, Stuart Warren, and P Wothers. *Organic Chemistry.* Oxford University Press, 2001.

[9] Connor W Coley, Regina Barzilay, Tommi S Jaakkola, William H Green, and Klavs F Jensen. Prediction of organic reaction outcomes using machine learning. *ACS Central Science*, 2017.

[10] Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *International Conference on Machine Learning*, pages 2702–2711, 2016.

[11] Hanjun Dai, Yingtao Tian, Bo Dai, Steven Skiena, and Le Song. Syntax-directed variational autoencoder for structured data. *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=SyqShMZRb`.

[12] Li Dong and Mirella Lapata. Language to logical form with neural attention. *arXiv preprint arXiv:1601.01280*, 2016.

[13] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232, 2015.

[14] Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A Smith. Recurrent neural network grammars. *arXiv preprint arXiv:1602.07776*, 2016.

[15] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017.

[16] Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 2016. doi: 10.1021/acscentsci.7b00572.

[17] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, volume 2, pages 729–734. IEEE, 2005.

[18] Gabriel Lima Guimaraes, Benjamin Sanchez-Lengeling, Pedro Luis Cunha Farias, and Alán Aspuru-Guzik. Objective-reinforced generative adversarial networks (organ) for sequence generation models. *arXiv preprint arXiv:1705.10843*, 2017.

[19] Markus Hartenfeller, Martin Eberle, Peter Meier, Cristina Nieto-Oberhuber, Karl-Heinz Altmann, Gisbert Schneider, Edgar Jacoby, and Steffen Renner. A collection of robust organic synthesis reactions for in silico molecule design. *Journal of chemical information and modeling*, 51(12):3093–3098, 2011.

[20] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

[21] David Janz, Jos van der Westhuizen, and José Miguel Hernández-Lobato. Actively learning what makes a discrete sequence valid. *arXiv preprint arXiv:1708.04465*, 2017.

[22] Wengong Jin, Connor Coley, Regina Barzilay, and Tommi Jaakkola. Predicting organic reaction outcomes with weisfeiler-lehman network. In *Advances in Neural Information Processing Systems*, pages 2604–2613, 2017.

[23] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. *arXiv preprint arXiv:1802.04364*, 2018.

[24] Matthew A Kayala, Chloé-Agathe Azencott, Jonathan H Chen, and Pierre Baldi. Learning to predict chemical reactions. *Journal of chemical information and modeling*, 51(9):2209–2222, 2011.

[25] Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. Molecular graph convolutions: moving beyond fingerprints. *Journal of computer-aided molecular design*, 30(8):595–608, 2016.

[26] Diederik P Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representation*, 2015.

[27] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[28] Eliyahu Kiperwasser and Yoav Goldberg. Easy-first dependency parsing with hierarchical tree lstms. *arXiv preprint arXiv:1603.00375*, 2016.

[29] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

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

[31] Greg Landrum. Rdkit: Open-source cheminformatics. *Online). http://www. rdkit. org. Accessed*, 3(04):2012, 2006.

[32] James Law, Zsolt Zsoldos, Aniko Simon, Darryl Reid, Yang Liu, Sing Yoong Khew, A Peter Johnson, Sarah Major, Robert A Wade, and Howard Y Ando. Route designer: a retrosynthetic analysis tool utilizing automated retrosynthetic rule generation. *J. Chem. Inf. Model.*, 49(3):593–602, 2009. ISSN 1549-9596.

[33] Tao Lei, Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Deriving neural architectures from sequence and graph kernels. In *Proceedings of 34th International Conference on Machine Learning (ICML)*, 2017.

[34] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[35] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. 2018. URL `https://openreview.net/forum?id=Hy1d-ebAb`.

[36] D. M. Lowe. Patent reaction extraction: downloads; `https://bitbucket.org/dan2097/patent-reaction-extraction/downloads`. 2014.

[37] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. *arXiv preprint arXiv:1611.08402*, 2016.

[38] Jonas Mueller, David Gifford, and Tommi Jaakkola. Sequence to better sequence: continuous revision of combinatorial structures. In *International Conference on Machine Learning*, pages 2536–2544, 2017.

[39] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International Conference on Machine Learning*, pages 2014–2023, 2016.

[40] Emilio Parisotto, Abdel-rahman Mohamed, Rishabh Singh, Lihong Li, Dengyong Zhou, and Pushmeet Kohli. Neuro-symbolic program synthesis. *arXiv preprint arXiv:1611.01855*, 2016.

[41] Matthias Rarey and J Scott Dixon. Feature trees: a new molecular similarity measure based on tree matching. *Journal of computer-aided molecular design*, 12(5):471–490, 1998.

[42] David Rogers and Mathew Hahn. Extended-connectivity fingerprints. *Journal of chemical information and modeling*, 50(5):742–754, 2010.

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

[44] Kristof Schütt, Pieter-Jan Kindermans, Huziel Enoc Sauceda Felix, Stefan Chmiela, Alexandre Tkatchenko, and Klaus-Robert Müller. Schnet: A continuous-filter convolutional neural network for modeling quantum interactions. In *Advances in Neural Information Processing Systems*, pages 992–1002, 2017.

[45] Marwin HS Segler and Mark P Waller. Neural-symbolic machine learning for retrosynthesis and reaction prediction. *Chemistry-A European Journal*, 2017.

[46] Marwin HS Segler, Thierry Kogej, Christian Tyrchan, and Mark P Waller. Generating focussed molecule libraries for drug discovery with recurrent neural networks. *arXiv preprint arXiv:1701.01329*, 2017.

[47] Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. *arXiv preprint arXiv:1802.03480*, 2018.

[48] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[49] Teague Sterling and John J Irwin. Zinc 15–ligand discovery for everyone. *J. Chem. Inf. Model*, 55(11):2324–2337, 2015.

[50] Sara Szymkuc, Ewa P. Gajewska, Tomasz Klucznik, Karol Molga, Piotr Dittwald, Micha Startek, Micha Bajczyk, and Bartosz A. Grzybowski. Computer-assisted synthetic planning: The end of the beginning. *Angew. Chem., Int. Ed.*, 55 (20):5904–5937, 2016. ISSN 1521-3773. doi: 10.1002/anie.201506101. URL `http://dx.doi.org/10.1002/anie.201506101`.

[51] Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

[52] Matthew H Todd. Computer-aided organic synthesis. *Chemical Society Reviews*, 34(3):247–266, 2005.

[53] Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. Grammar as a foreign language. In *Advances in Neural Information Processing Systems*, pages 2773–2781, 2015.

[54] Wendy A Warr. A short review of chemical reaction database systems, computer-aided synthesis design, reaction prediction and synthetic feasibility. *Molecular Informatics*, 33(6-7):469–476, 2014.

[55] Jennifer N Wei, David Duvenaud, and Alán Aspuru-Guzik. Neural networks for the prediction of organic chemistry reactions. *ACS Central Science*, 2(10): 725–732, 2016.

[56] David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of chemical information and computer sciences*, 28(1):31–36, 1988.

[57] Xiaodan Zhu, Parinaz Sobihani, and Hongyu Guo. Long short-term memory over recursive structures. In *International Conference on Machine Learning*, pages 1604–1612, 2015.

Appendix A

Molecular Graph Generation

A.1 Tree Decomposition

Algorithm 2 presents our tree decomposition of molecules. V_1 and V_2 contain non-ring bonds and simple rings respectively. Simple rings are extracted via RDKit's `GetSymmSSSR` function. We then merge rings that share three or more atoms as they form bridged compounds. We note that the junction tree of a molecule is not unique when its cluster graph contains cycles. This introduces additional uncertainty for our probabilistic modeling. To reduce such variation, for any of the three (or more) intersecting bonds, we add their intersecting atom as a cluster and remove the cycle connecting them in the cluster graph. Finally, we construct a junction tree as the maximum spanning tree of a cluster graph $(\mathcal{V}, \mathcal{E})$. Note that we assign an large weight over edges involving clusters in V_0 to ensure no edges in any cycles will be selected into the junction tree.

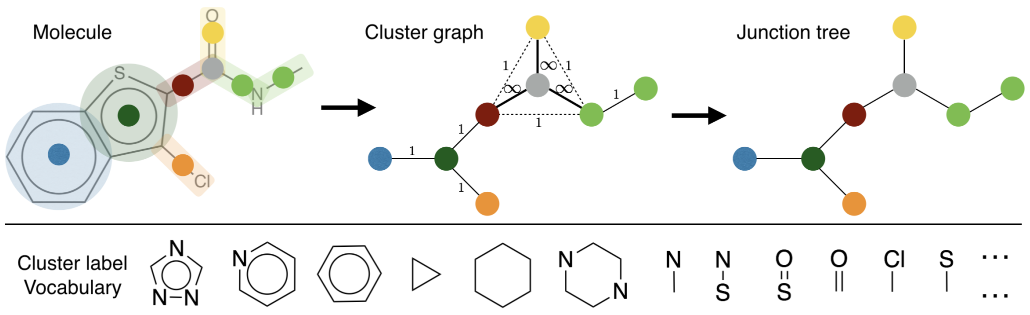


Figure A-1: Illustration of tree decomposition and sample of cluster label vocabulary.

Algorithm 2 Tree decomposition of molecule $G = (V, E)$

```
$V_1 \leftarrow$ the set of bonds $(u, v) \in E$ that do not belong to any rings.
$V_2 \leftarrow$ the set of simple rings of $G$.
for $r_1, r_2$ in $V_2$ do
    Merge rings $r_1, r_2$ into one ring if they share more than two atoms (bridged rings).
end for
$V_0 \leftarrow$ atoms being the intersection of three or more clusters in $V_1 \cup V_2$.
$\mathcal{V} \leftarrow V_0 \cup V_1 \cup V_2$
$\mathcal{E} \leftarrow \{(i, j, c) \in \mathcal{V} \times \mathcal{V} \times \mathbb{R} \mid |i \cap j| > 0\}$. Set $c = \infty$ if $i \in V_0$ or $j \in V_0$, and $c = 1$ otherwise.
Return The maximum spanning tree over cluster graph $(\mathcal{V}, \mathcal{E})$.
```

A.2 Stereochemistry

Though usually presented as two-dimensional graphs, molecules are three-dimensional objects, i.e. molecules are defined not only by its atom types and bond connections, but also the spatial configuration between atoms (chiral atoms and cis-trans isomerism). *Stereoisomers* are molecules that have the same 2D structure, but differ in the 3D orientations of their atoms in space. We note that stereochemical feasibility could not be simply encoded as context free or attribute grammars.

Empirically, we found it more efficient to predict the stereochemical configuration separately from the molecule generation. Specifically, the JT-VAE first generates the 2D structure of a molecule m, following the same procedure described in section 3.2. Then we generate all its stereoisomers $\mathcal{S}_m$ using RDKit's `EnumerateStereoisomers` function, which identifies atoms that could be chiral. For each isomer $m' \in \mathcal{S}_m$, we encode its graph representation $\mathbf{h}_{m'}$ with the graph encoder and compute their cosine similarity $f^s(m') = \cos(\mathbf{h}_{m'}, \mathbf{z}_m)$ (note that $\mathbf{z}_m$ is stochastic). We reconstruct the final 3D structure by picking the stereoisomer $\widehat{m} = \arg\max_{m'} f^s(m')$. Since on average only few atoms could have stereochemical variations, this post ranking process is very efficient. Combining this with tree and graph generation, the molecule reconstruction loss $\mathcal{L}$ becomes

$$\mathcal{L} = \mathcal{L}_c + \mathcal{L}_g + \mathcal{L}_s; \qquad \mathcal{L}_s = f^s(m) - \log \sum_{m' \in \mathcal{S}_m} \exp(f^s(m')) \tag{A.1}$$

A.3 Training Details

By applying tree decomposition over 240K molecules in ZINC dataset, we collected our vocabulary set $\mathcal{X}$ of size $|\mathcal{X}| = 780$. The hidden state dimension is 450 for all modules in JT-VAE and the latent bottleneck dimension is 56. For the graph encoder, the initial atom features include its atom type, degree, its formal charge and its chiral configuration. Bond feature is a concatenation of its bond type, whether the bond is in a ring, and its cis-trans configuration. For our tree encoder, we represent each cluster with a neural embedding vector, similar to word embedding for words. The tree and graph decoder use the same feature setting as encoders. The graph encoder and decoder runs three iterations of neural message passing. For fair comparison to SMILES based method, we minimized feature engineering. Our VAE loss function is $\mathcal{L} + \alpha\mathcal{L}_{KL}$ where $\mathcal{L}_{KL}$ is the KL-divergence. We found setting $\alpha = 1$ greatly hurts the training reconstruction performance (mostly for tree reconstruction), similarly reported in Kusner et al. [30], where they used $\alpha = 1/56$. We explore $\alpha = 0.001, 0.005$ and report the test result from setting with best validation performance. We use PyTorch to implement all neural components and RDKit to process molecules.

A.4 More Experimental Results

Sampled Molecules Note that a degenerate model could also achieve 100% prior validity by keep generating simple structures like chains. To prove that our model does not converge to such trivial solutions, we randomly sample and plot 250 molecules from prior distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$. As shown in Figure A-2, our sampled molecules present rich variety and structural complexity. This demonstrates the soundness of the prior validity improvement of our model.

Neighborhood Visualization Given a molecule, we follow Kusner et al. [30] to construct a grid visualization of its neighborhood. Specifically, we encode a molecule into the latent space and generate two random orthogonal unit vectors as two axis of a grid. Moving in combinations of these directions yields a set of latent vectors and we decode them into corresponding molecules. In Figure A-3 and A-4, we visualize the local neighborhood of two molecules presented in Dai et al. [11]. Figure A-3 visualizes the same molecule in Figure 3-4, but with wider neighborhood ranges.

Bayesian Optimization We directly used open sourced implementation in Kusner et al. [30] for Bayesian optimization (BO). Specifically, we train a sparse Gaussian process with 500 inducing points to predict properties of molecules. Five iterations of batch BO with expected improvement heuristic is used to propose new latent vectors. In each iteration, 50 latent vectors are proposed, from which molecules are decoded and added to the training set for next iteration. We perform 10 independent runs and aggregate results. In Figure A-5, we present the top 50 molecules found among 10 runs using JT-VAE. Following Kusner et al.'s implementation, the scores reported are **normalized** to zero mean and unit variance by the mean and variance computed from training set.

Constrained Optimization For this task, a property predictor F is trained jointly with VAE to predict $y(m) = logP(m) - SA(m)$ from the latent embedding of m. F is a feed-forward network with one hidden layer of dimension 450 followed by tanh activation. To optimize a molecule m, we start with its mean encoding $\mathbf{z}_m^0 = \boldsymbol{\mu}_m$ and apply 80 gradient ascent steps: $\mathbf{z}_m^t = \mathbf{z}_m^{t-1} + \alpha \frac{\partial y}{\partial z}$ with $\alpha = 2.0$. 80 molecules are decoded from latent vectors $\{\mathbf{z}_m^i\}$ and their property is calculated. Molecular similarity $sim(m, m')$ is calculated via Morgan fingerprint of radius 2 with Tanimoto similarity. For each molecule m, we report the best modified molecule m' with $sim(m, m') > \delta$ for some threshold δ. In Figure A-6, we present three groups of modification examples with $\delta = 0.2, 0.4, 0.6$. For each group, we present top three pairs that leads to best improvement $y(m') - y(m)$ as well as one pair decreased property ($y(m') < y(m)$). This is caused by inaccurate property prediction. From Figure A-6, we can see that tighter similarity constraint forces the model to preserve the original structure.

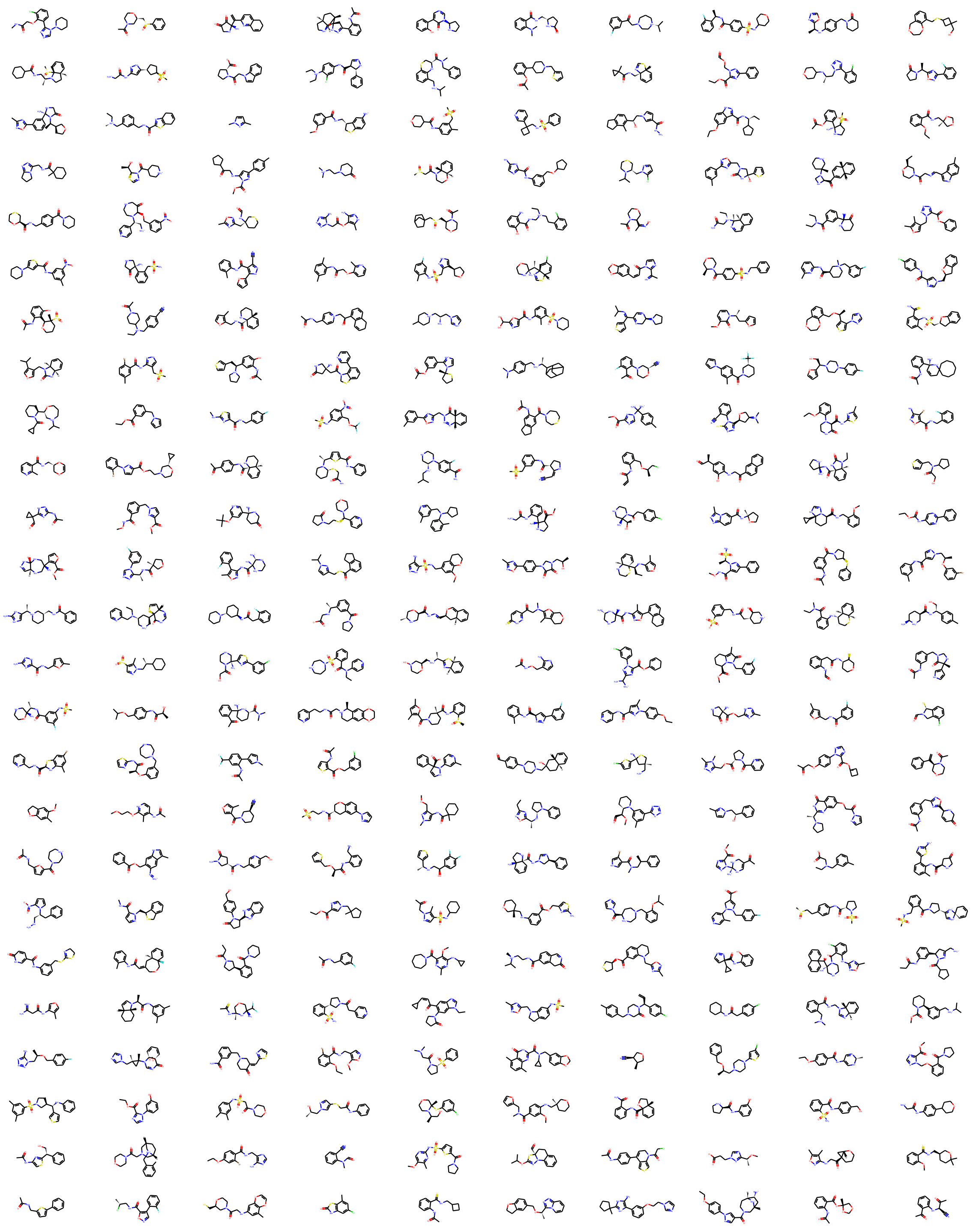


Figure A-2: 250 molecules sampled from prior distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$.

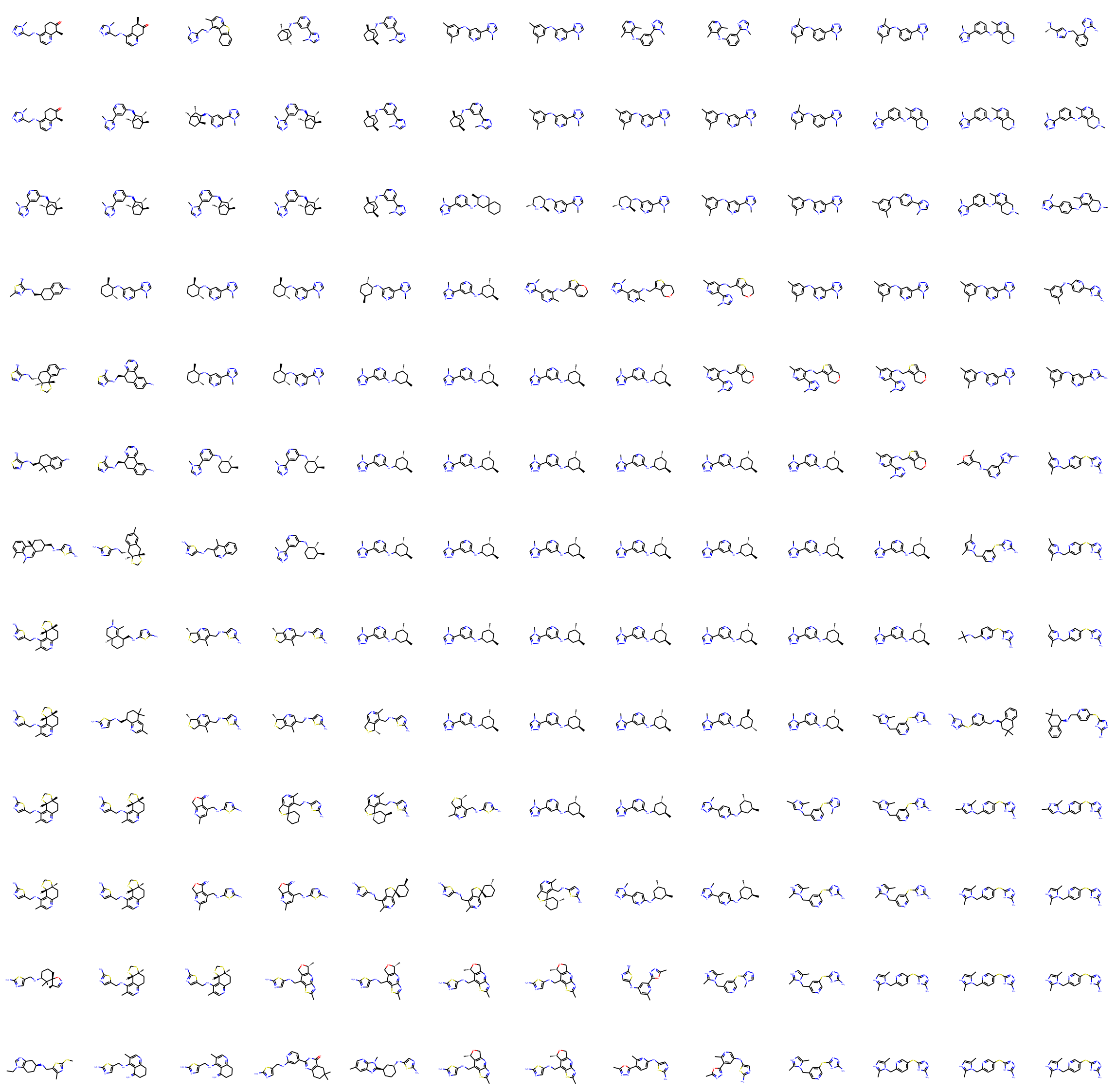


Figure A-3: Neighborhood visualization of molecule C[C@H]1CC(Nc2cncc(-c3nncn3C)c2)C[C@H](C)C1.

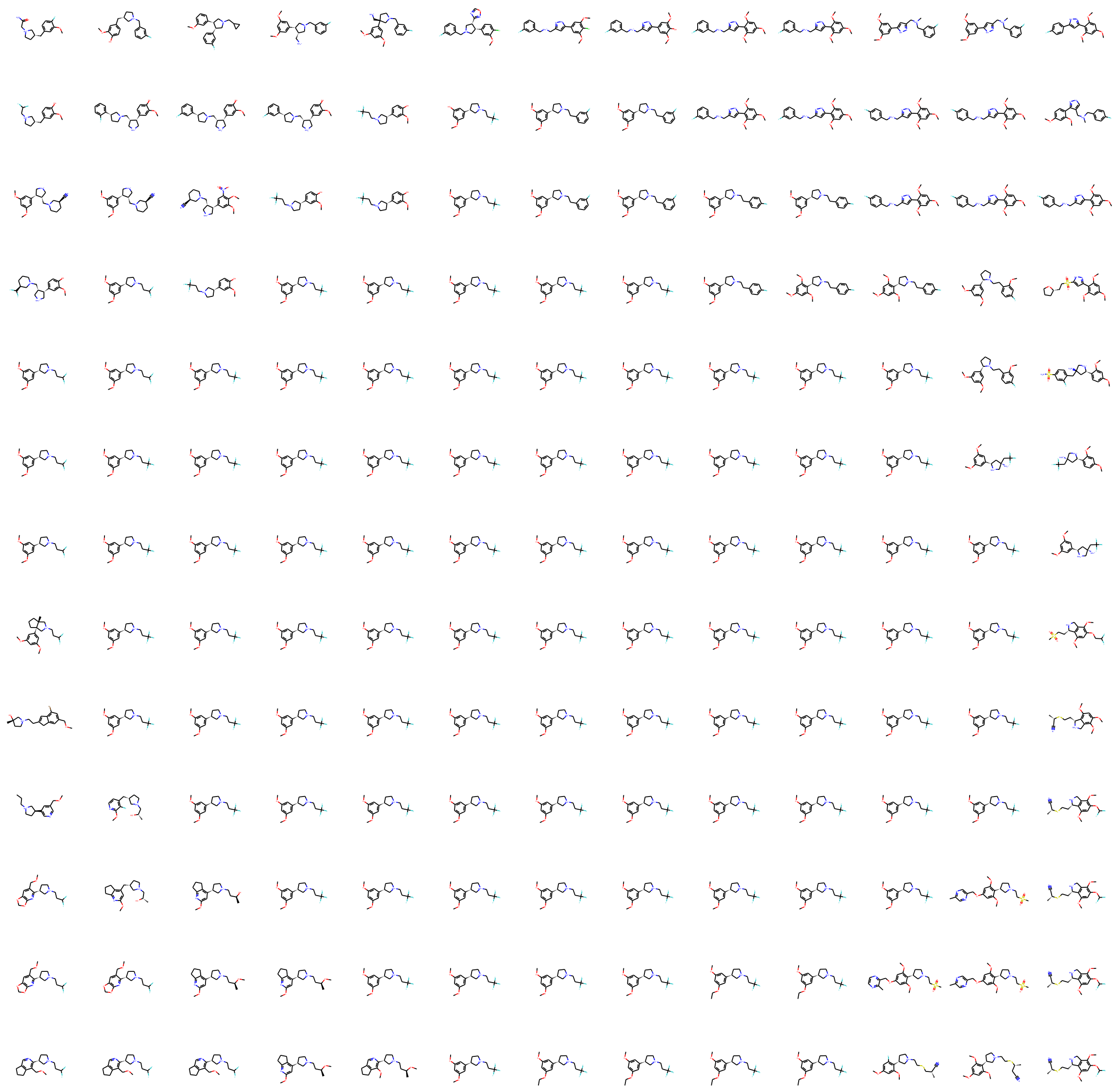


Figure A-4: Neighborhood visualization of molecule COc1cc(OC)cc([C@H]2CC[NH+](CCC(F)(F)F)C2)c1.

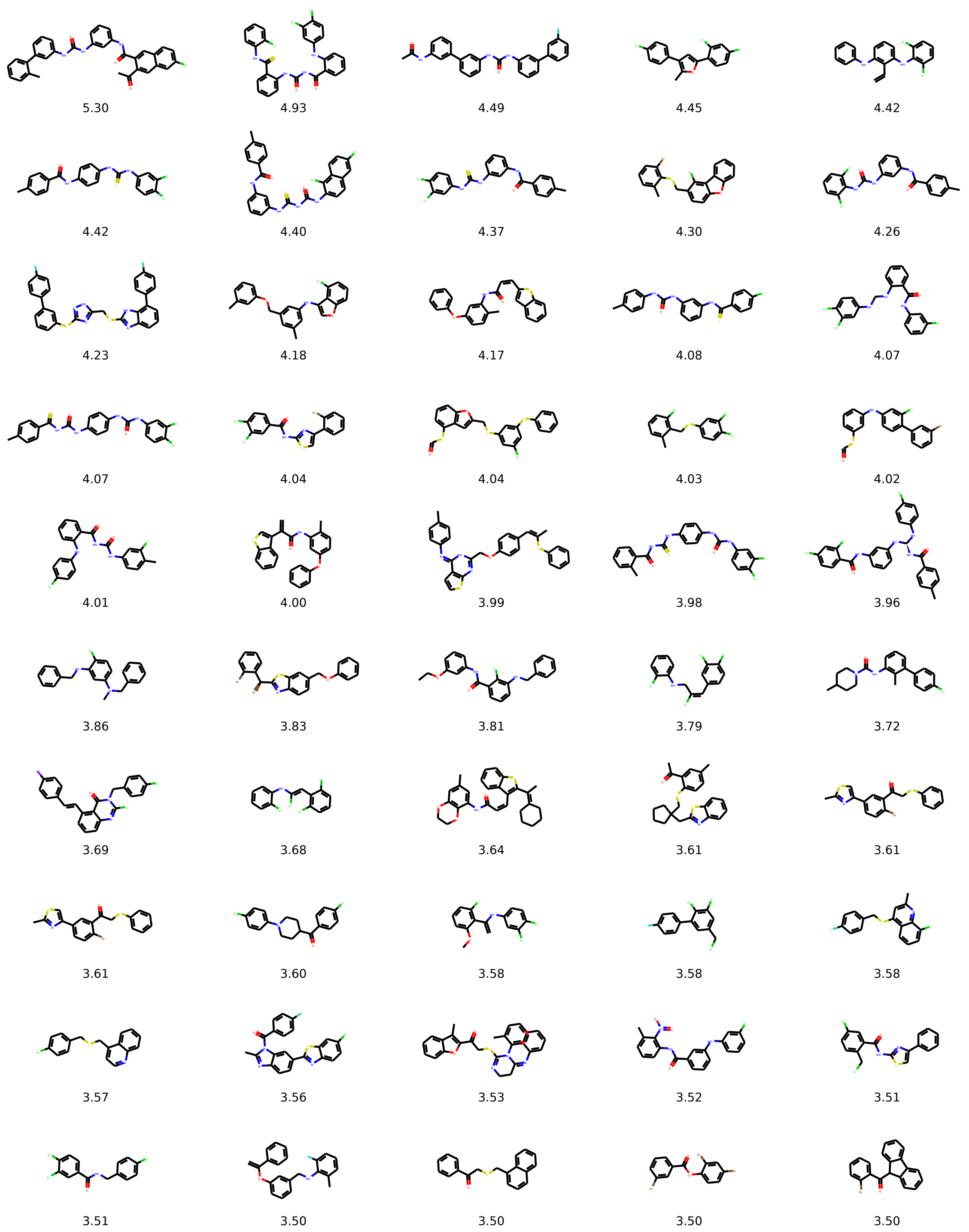


Figure A-5: Top 50 molecules found by Bayesian optimization using JT-VAE.

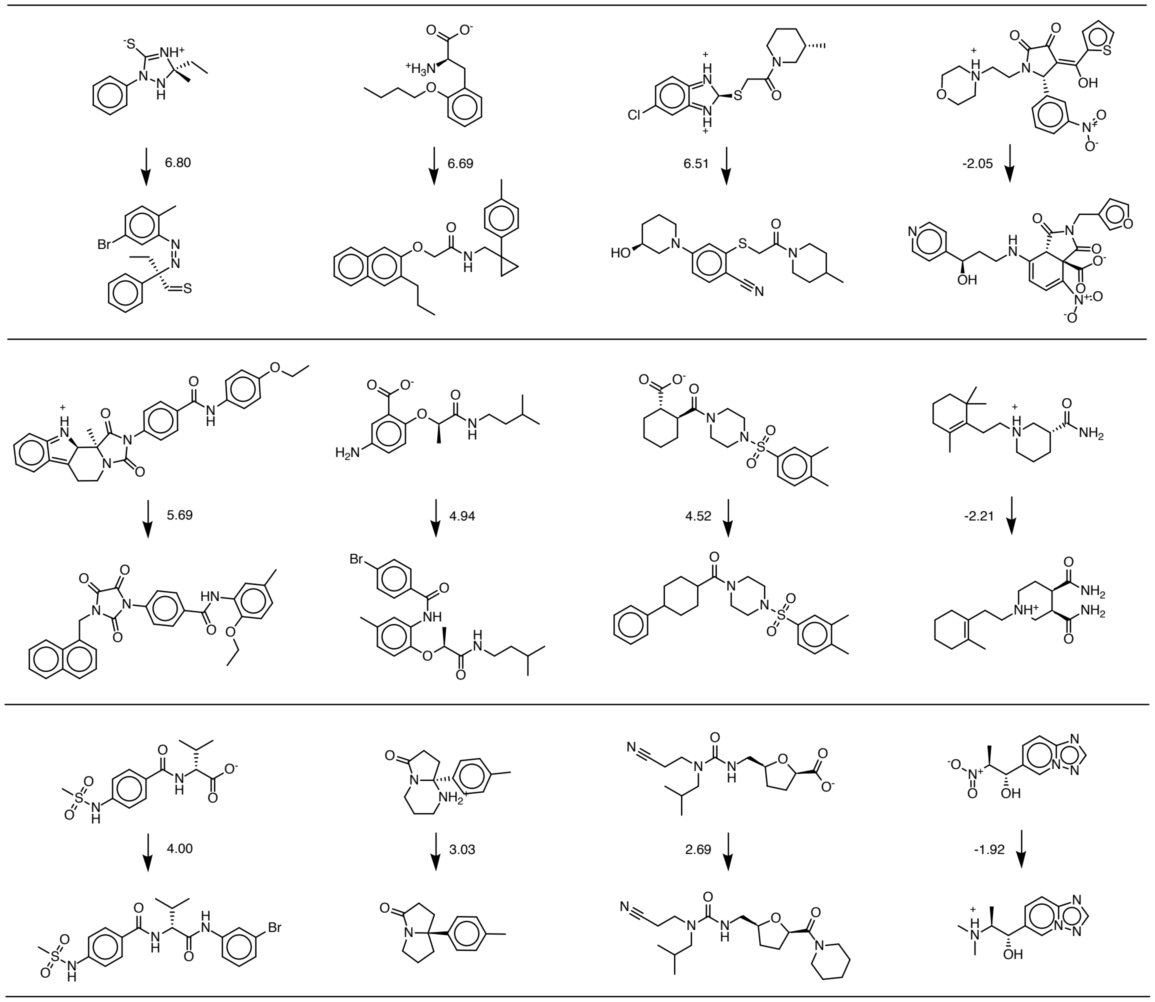


Figure A-6: Row 1-3: Molecule modification results with similarity constraint $sim(m, m') \geq 0.2, 0.4, 0.6$. For each group, we plot the top three pairs that leads to actual property improvement, and one pair with decreased property. We can see that tighter similarity constraint forces the model to preserve the original structure.