RetroGraph: Retrosynthetic Planning with Graph Search

Shufang Xie
Gaoling School of AI (GSAI)
Renmin University of China
shufangxie@ruc.edu.cn

Rui Yan*
Gaoling School of AI (GSAI)
Renmin University of China
ruiyan@ruc.edu.cn

Peng Han*
Aalborg University
pengh@cs.aau.dk

Yingce Xia
Microsoft Research Asia
yingce.xia@microsoft.com

Lijun Wu
Microsoft Research Asia
lijuwu@microsoft.com

Chenjuan Guo
East China Normal University
cjguo@dase.ecnu.edu.cn

Bin Yang
East China Normal University
byang@dase.ecnu.edu.cn

Tao Qin
Microsoft Research Asia
taoqin@microsoft.com

ABSTRACT

Retrosynthetic planning, which aims to find a reaction pathway to synthesize a target molecule, plays an important role in chemistry and drug discovery. This task is usually modeled as a search problem. Recently, data-driven methods have attracted many research interests and shown promising results for retrosynthetic planning. We observe that the same intermediate molecules are visited many times in the searching process, and they are usually independently treated in previous tree-based methods (e.g., AND-OR tree search, Monte Carlo tree search). Such redundancies make the search process inefficient. We propose a graph-based search policy that eliminates the redundant explorations of any intermediate molecules. As searching over a graph is more complicated than over a tree, we further adopt a graph neural network to guide the search over graphs. Meanwhile, our method can search a batch of targets together in the graph and remove the inter-target duplication in the tree-based search methods. Experimental results on two datasets demonstrate the effectiveness of our method. Especially on the widely used USPTO benchmark, we improve the search success rate to 99.47%, advancing previous state-of-the-art performance for 2.6 points.

CCS CONCEPTS

• **Applied computing** → **Computational biology**; **Molecular structural biology**.

KEYWORDS

retrosynthesis; retrosynthetic planning; graph neural network

*Corresponding author: Rui Yan (ruiyan@ruc.edu.cn), Peng Han (pengh@cs.aau.dk)

KDD '22, August 14–18, 2022, Washington, DC, USA.

ACM ISBN 978-1-4503-9385-0/22/08…$15.00
https://doi.org/10.1145/3534678.3539446

ACM Reference Format:
Shufang Xie, Rui Yan, Peng Han, Yingce Xia, Lijun Wu, Chenjuan Guo, Bin Yang, and Tao Qin. 2022. RetroGraph: Retrosynthetic Planning with Graph Search. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3534678.3539446

1 INTRODUCTION

Retrosynthetic planning aims to find pathways to synthesize novel molecules and is an important topic in chemistry and drug discovery. As shown in Figure 1, given a target molecule and a set of ingredient molecules, the goal is to find a pathway where the target can be eventually synthesized using ingredient molecules, and each step of the pathway is a viable chemical reaction. Such a process is usually modeled as a tree-search problem, where the starting point is the target molecule, and the path is the chemical reaction. Research focus [5, 17, 19] is how to find a feasible pathway with as few trials as possible.

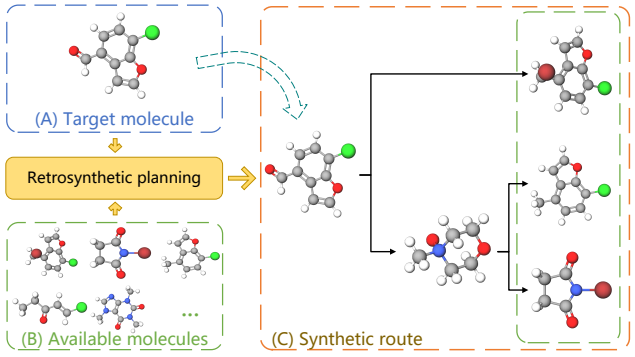

Figure 1: Example of retrosynthetic planning. Given a target molecule (A) and a set of available molecules (B), the retrosynthetic planning system is to find a feasible route (C).

Various data-driven search policies have been leveraged / proposed for this task, such as A* search [5], proof number search [22], and Monte Carlo tree search [17]. They treat the planning progress

as a tree expansion problem. However, as shown in Section 4.4, the success of those methods is limited in retrosynthetic planning due to intra-target and inter-target redundancy in search trees:

(1) The search tree of one target molecule could have multiple identical subtrees, because similar reactions can produce the same intermediate molecules. For example, known as Finkelstein Reaction [12], *iodoethane* can be synthesized by *bromoethane + podium iodide* or *bromoethane + potassium iodide*. Therefore, in the search tree of *iodoethane*, the whole subtree of *bromoethane* will be expanded generated multiple times in previous work [5, 19] , making the search inefficient, which is referred as *intra-target* redundancy.

(2) Previous methods [5, 19] treat different targets separately while ignoring that they may share some common intermediate molecules after several reactions, which is common in synthetic chemistry. For example, *6-aminopenicillanic acid (6-APA)* is a common intermediate for a lot of *β-lactam antibiotics* such as *benzylpenicillin*, *amoxycillin*, and *methicillin* [2]. Previous methods repeatedly visit the same intermediate molecules and their sub-trees while planning for related target molecules, which results in *inter-target* redundancy.

Therefore, we propose to merge duplicated molecule nodes in the search trees to eliminate the intra-target and inter-target redundancy. More specifically, instead of modeling the search process by a tree, we use a directed graph to represent the search process. The graph has two kinds of nodes: molecule nodes and reaction nodes. A molecule node represents an unique molecule which can be a target molecule, an intermediate reactant, or an ingredient molecule. The children of a molecule node are possible reactions that can synthesize the molecule in one step. The reaction node represents a reaction, and its children are its corresponding reactants.

A technical challenge of searching over a retrosynthetic planning tree is that the selection of one intermediate molecule is affected by its sibling sub-trees due to the existence of reaction nodes. In an extreme case, if one reactant cannot be synthesized from a given set of ingredients, we do not want to waste trials on all the reactions involving this reactant. The problem becomes more challenging when we change the search structure from tree to graph because each molecule can have multiple predecessors (i.e., a molecule is a reactant of multiple reactions). Although we can design heuristic rules to handle this extreme case, making the rules complete and optimal is difficult. Instead, we propose a learning-based method that uses a graph neural network (GNN) to guide search. This GNN takes a whole search graph as input and outputs a score for each molecular node in the graph to indicate the likelihood of the success of synthesizing a molecule. Because the whole graph information is available to the GNN, we expect it can better suggest the next node to expand.

Overall, the searching algorithm contains three steps: (a) selecting the most promising molecule to expand based on the known cost and GNN prediction of future cost; (b) expanding the selected node with a single step retrosynthetic network. Meanwhile, we merge duplicated molecule nodes in the search graph; (c) updating the nodes' properties (e.g., whether a node is success) for the next round. Within the exploration budget, we repeat such a process until we find retrosynthetic plans for all target molecules.

Furthermore, using the graph-search method, we can naturally process a batch of targets molecules together by building a hyper-graph that common ingredients are shared, i.e., there are no duplicated molecule nodes in the hyper-graph.

To demonstrate the effectiveness of our algorithm, we conduct extensive experiments on two datasets: the widely used benchmark dataset `USPTO` [5], and a new `USPTO-EXT` dataset which is ten times the size of `USPTO`. On the `USPTO` dataset, we achieved a 99.47% success rate under the constraint of 500 search steps constraint using single-target search, a new record better than previous results with 2.6 points. In addition, the plans also has better quality in terms of plan length. We also achieve a 72.89% success rate on the `USPTO-EXT` dataset on a single target search, which is better than baseline systems. We can future boost the success rate using batch target search. Finally, our study on the data shows that intra-target and inter-target duplication is critical for this task, and both graph search and GNN guidance contribute to performance improvements.

Our contribution can be summarized as follows:

- We propose a graph search method for retrosynthetic planning task that can eliminate the intra-target duplication in conventional tree search methods.
- We also propose a novel GNN guided policy based on the whole search graph to better select candidate nodes in the search graph.
- The proposed method can naturally search a batch of targets together by eliminating the inter-target redundancy. This can help further improve the success rate.

2 PRELIMINARIES

2.1 Retrosynthetic planning

Let $\mathcal{M}$ denote the space of all molecules, and let $\mathcal{I}$ denote the collections of available molecules, $\mathcal{I} \subseteq \mathcal{M}$. We have T target molecules to be synthesized, which are denoted as $\mathcal{T} = \{t_i\}_{i=1}^{T}, t_i \in \mathcal{M}$. The goal is to synthesize all molecules in $\mathcal{T}$ with the ingredients in $\mathcal{I}$.

Given any $m \in \mathcal{M}$, we might have n ways to synthesize it:

$$R(m) = \{R_1(m), R_2(m), \cdots, R_n(m)\}. \tag{1}$$

Each reaction $P_i(m)$ denotes triplet $(m, \mathcal{R}_i(m), c_i(m))$: m is the reaction product, $\mathcal{R}_i(m) \subset \mathcal{M}$ refers to the reactants, and $c_i(m) \in \mathbb{R}_+$ is the reaction cost. If $\exists R_i(m) \in R(m)$ satisfying that $\forall m' \in \mathcal{R}_i(m)$, $m' \in \mathcal{I}$, then we find a successful path way to synthesize m. Otherwise, we need to pick some $R_i(m)$ to further synthesize all the $t'' \in \mathcal{R}_i(m) \backslash \mathcal{I}$. If the elements in $\mathcal{R}_i(m) \backslash \mathcal{I}$ can be recursively synthesized using one or multiple steps with molecules in $\mathcal{I}$, then we successfully find a path too.

The mapping from m to $P(m)$ is realized by the single-step retrosynthesis model B with parameters θ_b,

$$B(\cdot; \theta_b) = m \mapsto \{R(m)_i = (m, \mathcal{R}_i, c_i)\}_{i=1}^{K}, \tag{2}$$

which generate at most K reactions for a given molecule m. Furthermore, there is a limitation of how many times the single-step model $B(\cdot; \theta_b)$ can be used.

2.2 A* search

A* search [15] is an efficient method for many planning tasks and have shown promising results in previous works [5, 19]. In A*

search, the total cost of a search state x is defined as

$$f(x) = g(x) + h(x), \tag{3}$$

where the total cost $f(x)$ is decomposed into two parts: the known cost $g(x)$ and the $h(v|G)$ the heuristic future cost estimation $h(x)$. Usually, the $g(x)$ is easy to define because all required information is available and the key challenge is to define an efficient $h(x)$.

3 PROPOSED METHOD

3.1 AND-OR search graph

Denote the search graph as $G = (\mathcal{V}, \mathcal{E})$, where the $\mathcal{V}$ is the node set and the $\mathcal{E}$ is the edge set. G is a directed graph, i.e., given an edge $e = (v_s, v_t)$, v_s is the predecessor and v_t is the successor. Different from a tree, each node might have multiple predecessors and successors.

There are two kinds of nodes in the graph G: the molecule node and the reaction node. Each node only has one type. Let $\mathcal{V}_m$ and $\mathcal{V}_r$ denote the collections of molecule nodes and reaction nodes. We also guarantee that $\mathcal{V}_m \cup \mathcal{V}_r = \mathcal{V}$ and $\mathcal{V}_m \cap \mathcal{V}_r = \emptyset$. There is no edge between the nodes of the same type. That is, molecule nodes can only be connected to reaction nodes, and vice versa. In other words, the types of a node and its predecessors/successors are different.

An example of the search graph is shown in Figure 2, where the M's and the R's are molecule nodes and reaction nodes respectively. The target $M0, M1 \in \mathcal{T}$ and the building blocks $M2, M9 \in \mathcal{I}$. $M0$ can be synthesized using reaction $R0$ *or* $R1$. If we choose $R0$, we use $M2$ *and* $M7$ to synthesize $M0$. If we choose $R1$, we use $M3$ *and* $M4$ to synthesize $M0$. Notably, the nodes $M4, M6, M7$, and, $M8$ have more than one predecessors, making the search graph different from a search tree.

Further, the $\mathcal{V}_m$ is split into two subsets: the open molecule nodes $\mathcal{V}_{\text{mo}}$ and closed molecule nodes $\mathcal{V}_{\text{mc}}$. A molecule node $v \in \mathcal{V}_{\text{mc}}$ only if the molecule in available set $\mathcal{I}$ or it has successors. Otherwise, the node belongs to $\mathcal{V}_{\text{mo}}$. For example, in Figure 2(A), $M3, M5, M6, M7, M8 \in \mathcal{V}_{\text{mo}}$ and other molecules are closed.

We define the Boolean function $\text{succ}(v) \mapsto \{\text{true}, \text{false}\}$ that evaluate the success state of each node. The search graph satisfy the AND-OR constraint [5]. Mathematically[1],

$$\text{succ}(v) = \begin{cases} \bigwedge\limits_{(v,v_k)\in\mathcal{E}} \{\text{succ}(v_k)\} & v \in \mathcal{V}_r \\ \bigvee\limits_{(v,v_k)\in\mathcal{E}} \{\text{succ}(v_k)\} \vee \{v \in \mathcal{I}\} & v \in \mathcal{V}_m \end{cases}, \tag{4}$$

where $\wedge, \vee$ denote logical AND, OR operation, respectively. Specifically,

(1) The reaction nodes are AND nodes. A reaction node is in true status if all its successors are in true status. Intuitively, a successful reaction node means all the reactants for this reaction are obtainable.

(2) The molecule nodes are OR nodes. A molecule node v is in true status if one of the following two constraints are satisfied: (i) the molecule v is in the set $\mathcal{I}$; (ii) it has at least one successor v_k, whose status is true. Intuitively, a successful molecule node means that it is already in the available set or we can synthesize v using the ingredients in $\mathcal{I}$.

One difference between a search tree and a search graph is that a graph may contain cycles. However, cycle avoidance is not easy for two reasons: First, detecting cycles in a graph is time-consuming because we need to visit all nodes to determine if a graph has a circle or not. Second, even after a cycle is detected, it is not trivial to decide which nodes and edges to remove to break the cycle. Fortunately, we notice that cycle will not affect the success checking in Equation (4). More specifically, for a node $v \in \mathcal{I}$, the molecule is already available, and no exploration of v is required. This means the out-degree of v is zero, ensuring that v is never in a cycle. From the recursive definition in Equation (4), we can see that all success routes must end at available molecules. Thus, the cycle dependency will not cause a fake success status. Therefore, we use a cycle-tolerance strategy in our planning algorithm.

[1]To simplify notations, we do not distinguish a node and the molecule/reaction it represents when there is no ambiguity.

Algorithm 1: RetroGraph Planning

Input: target molecules $\mathcal{T}$, single-step model $B(\cdot;\theta_b)$, available molecules $\mathcal{I}$, planning step budget N, single-step model candidate count K.

Initialize search graph $G = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \mathcal{T}$ and $\mathcal{E} = \emptyset$;
repeat
  /* Select the next molecule node v_{next} to expand using the Equation (10) */
  $v_{next} \leftarrow \arg\min_{v\in\mathcal{V}_{mo}} \text{cost}(v|G)$;
  /* Compute top K reactions by model $B(\cdot;\theta_b)$ */
  $\{R_i(v_{\text{next}})\}_{i=1}^{K} \leftarrow B(v_{\text{next}};\theta_b)$;
  $\Upsilon \leftarrow \emptyset$;
  for $i = 1$ ***to*** K **do**
    Expand G by $\{R_i(v_{\text{next}})\}_{i=1}^{K}$ using Equation (6, 7);
    Update the affected nodes to Υ;
  end
  for $v \in \Upsilon$ **do**
    Bottom-up update status of G from v;
  end
  $N \leftarrow N - 1$;
until $N = 0$ *or* $succ(t) = true : \forall t \in \mathcal{T}$ *or* $|\mathcal{V}_{mo}| = 0$;

3.2 Planning procedure

Our planning algorithm contains three steps and details are shown in Algorithm 1.

Step A: Selection. Given a search graph G, the next node to further explore is

$$v_{next} = \arg\min_{v\in\mathcal{V}_{mo}} \text{cost}(v|G), \tag{5}$$

where cost refers to the estimated cost of exploring v conditioned on the existing graph G. A good cost should consider both existing cost of obtaining v_{next} and its future cost. We will introduce more details about cost is Section 3.3.

Step B: Expansion. This step expand G with v_{next}. Following [5, 19], we first extract the molecule features (i.e., Morgan fingerprint) of v_{next}, and then feed it into the single-step model $B(:, \theta_b)$. The

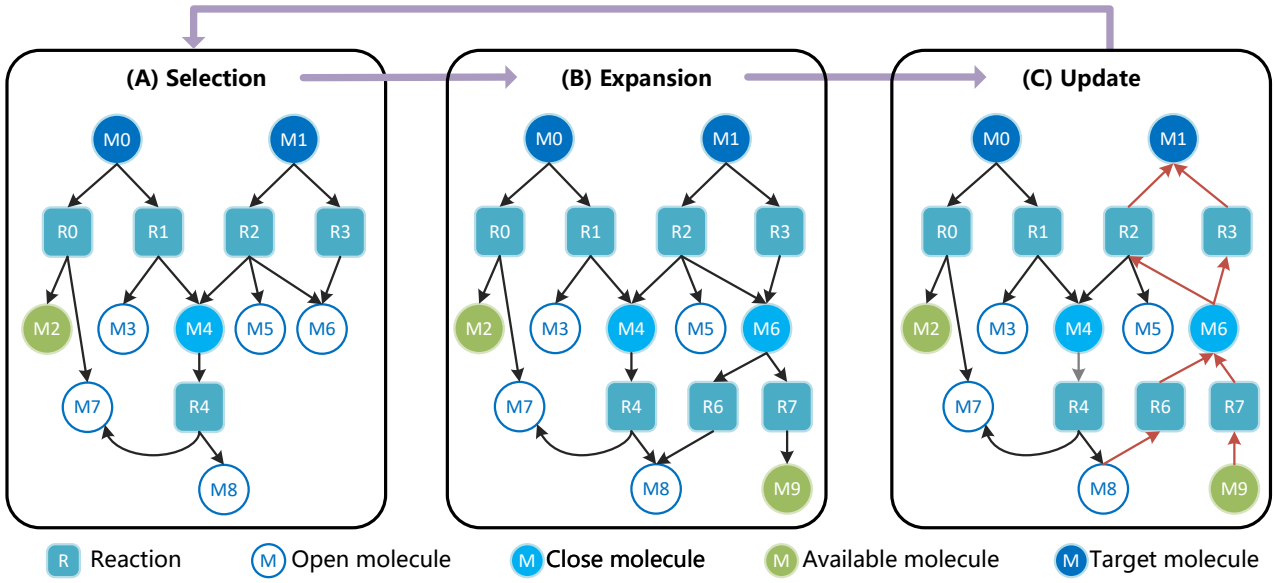

Figure 2: Overview of RetroGraph algorithm. The circles and squares denote molecule and reaction nodes, respectively. The molecule `M6` is selected for expansion in this round. The dark arrows are synthetic dependencies and the red arrows are bottom-up update process from `M8` and `M9`. Note the molecule `M4`, `M6`, `M7`, and `M8` are shared by multiple reactions.

single-step model then predicts the top K reactions[2]. These reactions are then applied to v_{next}, and we obtain the reactants and the corresponding costs. Finally, we update the search graph $G(\mathcal{V}, \mathcal{E})$ with reactions $\{R_i = (v_{\text{next}}, \mathcal{R}(v_{\text{next}})_i, c(v_{\text{next}})_i)\}_{i=1}^{K}$

$$\mathcal{V} \leftarrow \mathcal{V} \cup \{m_{i,j} | 1 \le i \le K, m_j \in \mathcal{R}_i(v_{\text{next}})\}, \tag{6}$$

$$\mathcal{E} \leftarrow \mathcal{E} \cup \{e(v_{v_{\text{next}}}, R_i)\}_{i=1}^{K} \cup \{e(R_i, m_j) | 1 \le i \le K, m_j \in \mathcal{R}_i(v_{\text{next}})\}. \tag{7}$$

An important property of this method is that no duplicated molecules are allowed in the graph. Therefore, we leverage a global molecule memory to store all molecules that already exists in the graph and void creating new one when it is already exits.

Step C: Update. After graph expansion,we need update the status (e.g., success state, historical cost) of all nodes. Here we use a bottom-up strategy and only update the affected nodes to save search time. More specifically, Let Υ denotes the nodes need to be updated, which is initialized by the molecules directly affected in Step B. For each molecule node $v \in \Upsilon$, we find all edges $e(v_p, v)$ that ends at v. Then for each v_p, we recompute success state and historical cost. We keep this process until all required nodes are updated.

We repeat these three steps until the termination condition is stratified. i.e, we have successfully found routes for all targets, used up all iteration budgets, or there are no nodes to expand. After this search process, we can extract synthetic routes as tree from the search graph because the tree structure synthetic routes are more accessible to humans. Therefore, we apply a depth-first iteration strategy on a success search graph. To be more specific, we start from a target molecule and select all success reactions of it. Then, we recursively visit the reactants until we meet the leaf nodes. We can build the synthetic tree from bottom to top during the recursive visit process.

[2]More specifically, it predicts the top K reaction templates. All templates are validated by Rdchiral [7], a RDKit wrapper, to remove the chemical impossible ones.

3.3 GNN guided policy

For the retrosynthetic planning, we can write the A* cost function in Equation (3) as

$$\text{cost}(v|G) = g(v|G) + h(v|G), \tag{8}$$

where the $g(v|G)$ is the known (historical) cost from the start point (a target node t in this task) to v and the $h(v|G)$ is the heuristic function that estimate the future cost.

Let $\Psi(v_1, v_2, \cdots, v_n)$ denotes a path of graph $G(\mathcal{V}, \mathcal{E})$ where $v_i \in \mathcal{V} : \forall v_i \in \Psi$ and $e(v_i, v_{i+1}) \in \mathcal{E} : \forall v_i \in \Psi (i < n)$. The history cost is defined as

$$g(v|G) = \min_{t \in \mathcal{T}} \min_{\Psi(v_t, \cdots, v)} \sum_{v_i \in \Psi \cap \mathcal{V}_r} c_i(v), \tag{9}$$

This means we select a target $t \in \mathcal{T}$ and a path $\Psi(t, \cdots, v)$ from t to v to compute a lower-bound of the history cost.

However, defining the $h(v|G)$ on an AND-OR search graph is more complicated. Because of the reaction nodes (AND nodes), the total route cost of a given molecule is related to itself and other molecules in the route. Therefore, complex computation in G is required to if we want to have a high-quality estimation of $h(v|G)$. Instead, we propose a learning-based method and use a GNN to guide the search. More specifically, we modify the Equation (8)

from the original form to

$$\text{cost}(v|G) = g(v|G) + \text{GNN}(v|G;\theta_G), \tag{10}$$

where the $\text{GNN}(v|G;\theta_G)$ is a GNN with parameter θ_G to guide the search. The GNN architecture is in Figure 3.

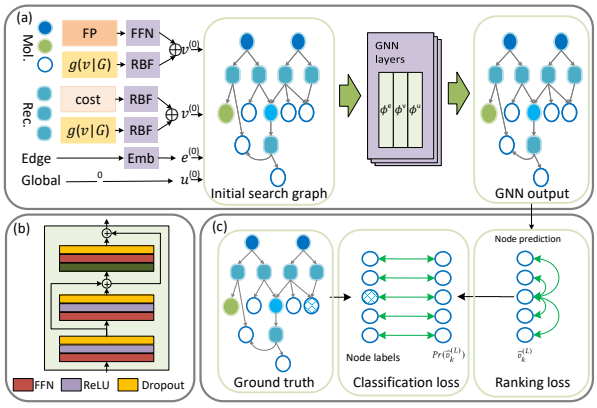

Figure 3: Illustration of policy GNN. (a) Graph representation and GNN process; (b) Illustration of MLP in Equation (19,20,21); (c) Computation of classification loss and ranking loss.

GNN architecture. Our GNN consists three components: node and edge embeddings, GNN layers, and output layer. First, we embed the node and edge features into the hidden representations. For molecule nodes, we use the historical cost and Morgan Fingerprint (FP) as the node feature, and for reaction nodes, we use the historical cost and reaction cost as node feature. Meanwhile, we use different edge embedding for different directions of edges as edge feature. Finally, we use a zero vector to initialize the global (i.e., graph level) representation. The FP is project to the hidden dimension using a feedforwad (FFN) layer. All scale values (i.e., historical cost, reaction cost) is embedded by the Radial Basis Function (RBF) kernel, which is defined as:

$$\text{RBF}(x;L,H,N)_i = \exp\left(\frac{-\left(x - i * \frac{H-L}{N}\right)^2}{\tau}\right),\ 0 \le i < N, \tag{11}$$

where L, H, N are the lowerbound, upperbond, and embedding size, respectively. Intuitively, this function embed a number $x \in \mathbb{R}$ using a series of Gaussian distributions. If we denote the node, edge, and global representation at $i-$th layer $v^{(i)}, e^{(i)}, u^{(i)}$, respectively, the initialization process can be written as:

$$e_k^{(0)} = \text{embed}(\text{dir}(e_k)), \tag{12}$$

$$v_k^{(0)} = \text{RBF}\left(\text{cost}_h(v)\right) \oplus \left(\text{FFN}(\text{FP}(v_k)) :\ v_k \in \mathcal{V}_m, \right. \tag{13}$$

$$v_k^{(0)} = \text{RBF}\left(\text{cost}_h(v)\right) \oplus \left(\text{RBF}(c_k)\right) :\ v_k \in \mathcal{V}_r, \tag{14}$$

$$u^{(0)} = 0. \tag{15}$$

To model both the global and local structure of the search graph, we use meta GNN Layer [3] as the building layers of the policy GNN. The meta GNN layer consists of three sub-layers that update the edge, node, and global representations. Suppose we have L GNN layers, for $1 \le i \le L$, the update process is

$$e_k^{(i)} = \phi^e\left(e_k^{(i-1)}, v_{sk}^{(i-1)}, v_{kt}^{(i-1)}, u^{(i-1)}\right), \tag{16}$$

$$v_k^{(i)} = \phi^v\left(\mathrm{e}_k^{(i)}, v_k^{(i-1)}, u^{(i-1)}\right), \tag{17}$$

$$u^{(i)} = \phi^u\left(e^{(i)}, v^{(i)}, u^{(i-1)}\right). \tag{18}$$

Where $v_{\text{sk}}, v_{\text{kt}}$ means the start and end node of edge e_k and e_k denotes all edges e end at v_k. In our method, we use:

$$\phi^e = \text{MLP}\left(e_k^{(i-1)} \oplus v_{sk}^{(i-1)} \oplus v_{kt}^{(i-1)} \oplus u^{(i-1)}\right), \tag{19}$$

$$\phi^v = \text{MLP}\left(v_k^{(i-1)} \oplus \text{msg}^{(i)}(v_k) \oplus u^{(i-1)}\right), \tag{20}$$

$$\phi^u = \text{MLP}\left(u^{(i-1)} \oplus \frac{1}{N}\sum v_k^{(i)}\right), \tag{21}$$

where

$$\text{msg}^{(i)}(v_k) = \frac{1}{|\mathrm{e}_k|}\sum_{e_{sk}=(v_s,v_k)\in \mathrm{e}_k} \text{MLP}\left(v_s^{(i-1)} \oplus e_{sk}^{(i-1)}\right). \tag{22}$$

The $\oplus$ denotes the tensor concatenation operation and MLP denotes three-layer perception networks using ReLU activation [1], residual connection [16], and dropout [34] and the details are in Figure 3.

Finally, we use a feedforwad layer on each node to transform the node hidden representation in the last GNN layer $v_k^{(L)}$ to a scalar value $\widehat{v}_k^{(L)}$ and use the sigmoid function to convert v_k to probability $Pr(\widehat{v}_k^{(L)})$ for classification loss. During search time, we normalize all open nodes prediction to get the score for selection,

$$\text{GNN}(v_k|G;\theta_G) = \frac{\exp\left(\widehat{v}_k^{(L)}\right)}{\sum_{v\in\mathcal{V}_{mo}}\exp\left(\widehat{v}^{(L)}\right)}. \tag{23}$$

GNN training We use an offline-training strategy to train the GNN policy network. More specifically, we first use A* search on the training dataset to find synthetic plans for all target molecules. Next, we gradually follow the generated reactions for each successful plan to expand the search graph. We collect the graph representations as GNN input and the correct nodes to expand as GNN labels for each expansion step. We mark the correct nodes for expanding with positive labels and other open nodes with negative labels.

We treat the GNN training as both node classification and ranking tasks. First, we have a binary classification cross-entropy (CE) loss for each node because there could be more than one positive label. The loss function is defined as

$$\mathcal{L}_{bce} = -\frac{1}{|\mathcal{V}_{\text{mo}}|}\sum_{v\in\mathcal{V}_{\text{mo}}} y\log(Pr(\widehat{v})) - (1-y)\log(1-Pr(\widehat{v})). \tag{24}$$

Following Chen et al. [5], we also use a ranking loss that the $\widehat{v}$ of positive nodes are larger than the negative nodes with at least τ margin.

$$\mathcal{L}_{rank} = -\frac{1}{|\mathcal{V}_{op}|}\frac{1}{|\mathcal{V}_{on}|}\sum_{v_p\in\mathcal{V}_{op}}\sum_{v_n\in\mathcal{V}_{on}}\min(0,\widehat{v}_p - \widehat{v}_n - \tau). \tag{25}$$

where $\mathcal{V}_{op}, \mathcal{V}_{on}$ are positive and negative open nodes set, respectively. Therefore, the final loss of GNN is

$$\mathcal{L}_{\text{GNN}} = \mathcal{L}_{bce} + \mathcal{L}_{rank}. \tag{26}$$

In all our experiments, we use $\tau = 4$ and do not tune the weights of these two loss functions.

4 EXPERIMENTS

We present experimental results to answer the following questions:
Q1 (Section 4.2): How does our algorithm perform in single target search when eliminating intra-target redundancy?
Q2 (Section 4.3): How does our algorithm perform in batched target search when eliminating inter- and intra-target redundancy?
Q3 (Section 4.4): How frequently do inter-target and intra-target occur in tree search?
Q4 (Section 4.5): How do the graph search and the GNN policy network contribute to our system?

4.1 Experimental Setup

Dataset. We use two datasets to evaluate our method. The first one is a widely used benchmark USPTO dataset that was introduced by Retro* [5]. Considering that the scale of test data is small, we created an extra test set named USPTO-EXT that contains 10× sizes of test routes of USPTO. For single-step model training, we use the $1.3M$ reactions from Chen et al. [5] and follow [19] to create augmented training data using forward and backward models. For GNN training data, we follow the steps of Section 3.3 and collected about $140k$ graphs. We then randomly select $1,000$ graphs for the validation and $1,000$ graphs for the test and use the remaining graphs for training. Meanwhile, we use the same set of commercially available molecules set as [5, 19]. More details are in the Appendix.
Single-step model. Although our planning framework is generally compatible with any single-step model, we use the same model as Chen et al. [5], Kim et al. [19] to make a fair comparison with previous work. It is a template-based single-step retrosynthesis model that is introduced by Segler et al. [28], We use the same reaction template set as Chen et al. [5], which contains about $380k$ templates extracted from the original USPTO dataset using Rdchiral [7]. Meanwhile, the model architecture is a 2-layer MLP using the Morgan fingerprint [26] as input and the probability of using each template as output. It means it treats the single-step retrosynthetic as a multi-class classification problem. The fingerprints are extracted with radius 2 and 2048 bits. We use the top 50 predicted templates to expand the input molecule during planning. We use the same hyper-parameters as Kim et al. [19] to train the MLP model, with a learning rate of 0.001, dropout rate 0.4, and batch size 1024. The models are trained for 20 epochs using Adam optimizer [21].
Policy GNN. We use three Meta GNN layers [3] as the building blocks of policy GNN with hidden dimensions as 128. For RBF kernel, we use $L = 0, H = 10, N = 64$, and $\tau = \frac{(H-L)^2}{4}$ in all our experiments. The model is trained with Adam optimizer [21] using a learning rate of 0.0001 and dropout 0.1. We use 32 graphs per mini-batch and train the network with a max epoch of 20. The best checkpoint is selected by the ranking loss on the validation set.
Evaluation. We use USPTO and USPTO-EXT datasets to evaluate our method and compare it with previous works. More specifically, we measure the success rate with different iteration limits. For USPTO, we follow previous works to use 500 as the max iteration limit, and for USPTO-EXT, we use 100 to make the task more challenging. In addition to the success rate, we also show the results of average iteration, the average number for molecule nodes, the average number of reaction nodes under the limit of max iterations. Finally, we also compute the average route length and average route cost to evaluate the plan quality.
Baselines. We compare our system with representative baselines on this task and the details are available in Appendix. Briefly, **Greedy DFS** [17] is a classic planning method that always prioritizes the node with max probability; **DFPN-E** [22] is a Depth-First Proof-Number search method; **MCTS-rollout** [17], **EG-MCTS** [17], and **EG-MCTS-0** [17] are MCTS methods; **Retro*** [5], **Retro*-0** [5], **Retro*+** [19], and **Retro*+-0** [19] are A* search methods.

4.2 Single target planning

When the $|\mathcal{T}| = 1$, our method can search one target at a time. Therefore, we can make a fair comparison with the existing single target planning method. The results for USPTO and USPTO-EXT are shown in Table 1 and Table 2, respectively. Under all iteration limits, the success rates of our method are much better than baseline results. More specifically, when the iteration limit is 500, we achieved a 99.47% success rate, which is 2.63-points better than the previous state-of-the-art success rate. Meanwhile, the average iteration of our method is only 45.13, which is 10.7-point lower than EG-MCTS. For the average number of reaction and molecule nodes, we achieved 674.22 and 500.43, respectively. The average number of molecule nodes is only about half of the previous results because redundant molecule nodes are eliminated. These numbers demonstrate the correctness of our motivation and effectiveness of this method. In Table 3, we show the result of plan quality. Compared with baselines, we achieved lower route length and cost. Our average route length is 6.33, better than previous best results. Meanwhile, the route cost is better than most baselines and comparable with the best result. These results exhibit the effectiveness of our method.

4.3 Batch targets planning

One advantage of our method is that we can search a batch of target molecules together, which is not supported by tree search because no nodes are shared. To evaluate the effectiveness of batch targets planning, on USPTO , we use K-Means [25] with Morgan fingerprint to cluster all test molecules into 32 clusters and make mini-batches inside each cluster with different batch sizes. The results are in Figure 4 where each sub-figure represent different iteration limits. Meanwhile, the number of stars inside the bar represents the number of targets in a batch To make the problem more challenging, we reduce the max iteration budget to 100. From the results, we have the following findings:
• Under all iteration limits, our method (purple bars), regardless of using single target or batch targets planning, has better performance than conventional tree-based methods.
• Under all iteration limits, the batch targets planning have better performance than all tree-based and graph-based single-target search baselines.

Table 1: Experimental results on USPTO dataset with single target search. We compare each algorithm at the success rate of different limit. Under the limit of 500, we also show the average number of iterations , reaction (Rec.) nodes, and molecule (Mol.) nodes. The best results are marked as bold.

Algorithm	Success Rate of Iteration Limit [%] ↑					# Iteration ↓	# Rec. Nodes↓	# Mol. Nodes↓
	100	200	300	400	500			
Greedy DFS	38.42	40.53	44.21	45.26	46.84	300.56	-	-
DFPN-E	50.53	58.42	64.21	68.42	75.26	208.12	3123.33	4635.08
MCTS-rollout	43.68	47.37	54.74	58.95	62.63	254.32	-	-
Retro*-0	36.84	59.47	68.95	74.74	79.47	210.49	3908.15	5575.97
Retro*	52.11	66.32	76.84	81.05	86.84	166.72	2927.92	4174.52
Retro*+-0	67.37	82.10	93.16	95.26	96.32	96.14	1421.90	2108.50
Retro*+	71.05	85.26	88.95	90.00	91.05	100.15	1209.79	1767.81
EG-MCTS-0	57.37	63.68	68.42	71.05	73.68	186.15	2525.20	3339.52
EG-MCTS	85.79	92.63	94.21	95.79	96.84	55.84	869.59	1193.79
RetroGrph (Ours)	**88.42**	**97.89**	**98.95**	**99.47**	**99.47**	**45.13**	**674.22**	**500.43**

Table 2: Experimental results on USPTO-EXT dataset with single target search. We compare each algorithm at the success rate of different iteration limit. Under the limit of 100, we also show the average number of iterations, reaction (Rec.) nodes, and molecule (Mol.) nodes. The best results are marked as bold.

Algorithm	Success Rate of Iteration Limit [%] ↑						# Iteration ↓	# Rec. Nodes↓	# Mol. Nodes↓
	10	20	30	40	50	100			
Retro*-0	42.11	48.11	50.63	52.16	53.11	56.68	49.81	836.54	1198.50
Retro*	42.47	48.79	51.84	53.63	55.00	57.89	48.49	790.49	1136.51
Retro*+-0	48.74	54.11	57.21	58.68	60.42	66.16	42.55	535.04	806.52
Retro*+	49.05	55.00	59.11	61.42	63.63	68.74	40.11	469.42	716.10
RetroGraph (Ours)	**50.84**	**58.05**	**62.05**	**64.26**	**66.89**	**72.89**	**37.25**	**491.97**	**373.80**

Table 3: Experimental results of plan quality on USPTO.

Algorithm	Route Length	Route Cost
Retro*-0	11.21	19.40
Retro*	9.71	15.33
Retro*+-0	7.69	**11.66**
Retro*+	8.74	15.23
RetroGraph (Ours)	**6.33**	12.92

• With the increase of batch size, the performance will improve, but the performance will be saturated. It will be interesting to study the larger batch in future work.

4.4 Study of duplication in tree search

To study the intra-target redundancy, we use tree-based Retro* and Retro*+ on the USPTO test data and the results are shown in Figure 5. The horizontal axis is the total number of expanded nodes and the vertical axis is the number of unique molecules visited. The dash lines are the linear regression of data of each model. We can see that the regression line is much lower than the diagonal line, showing that intra-target redundancy is prevalent in the data. On the contrary, our graph search method does not have redundancy, and the data points will lie on the diagonal line.

To study the inter-target redundancy, we take the synthetic plans in the USPTO training set as oracle plans. Then we collect all intermediate reactants in oracle plans and count the number of times each reactant occurs in the plans. We observe that the distribution is long-tail. The most popular reactant *acetyl acetate* occurs in $2,989$ routes, which is nearly 1% of all oracle routes. Furthermore, on average, each reactant occurs 1.77 times in the routes, showing that inter-target redundancy is common.

The above analysis shows that intra-target and inter-target duplication are common issues in this task. Solving this issue is critical for performance.

4.5 Ablation study

We conducted an ablation study on the USPTO dataset, and the results is available in Figure 6 where the x-axis denotes the iteration limit, and the y-axis denotes the success rate. To study the effect of our GNN policy network, we keep the graph search structure and replace the GNN with the value network used by Retro* and Retro*+. The value network is only based on single-molecule fingerprints and have no access to the complete search graph. We denote this system as RetroGraph-N. We also use slashes to mark the background of tree-based methods. From the results, we can observe that:

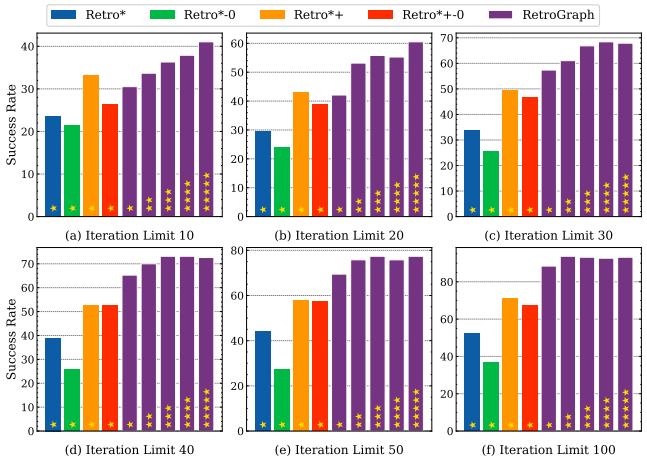

Figure 4: Batch targets planning on USPTO dataset. The number of starts in a bar denotes the number of targets.

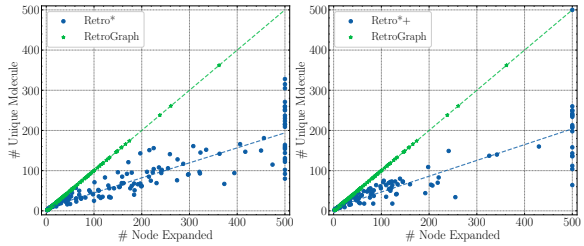

Figure 5: Study of intra-duplication in tree search. The left and right plots are results of Retro* and Retro*+, respectively.

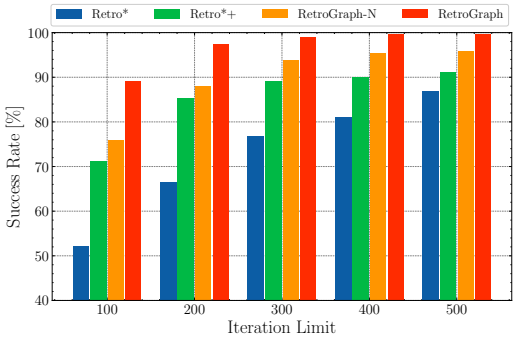

Figure 6: Ablation study on the USPTO dataset.

• Comparing Retro*, Retro*+, and RetroGraph-N, the three models that use identical value model checkpoint, the RetroGraph-N has the best performance, indicating that graph search is critical.
• The graph-based search methods (i.e., the RetroGraph and the RetroGraph-N) perform better than the tree-based methods (i.e., Retro* and Retro*+), demonstrating that graph search is effective.
• Among graph-based methods, the RetroGraph has better performance than RetroGraph-N, showing that guiding the search by GNN is also essential for this task.

5 RELATED WORK

5.1 Retrosynthetic planning

The data-driven retrosynthetic planning have attracted many research attention [5, 9, 13, 14, 17–19, 27, 29, 35]. Chen et al. [5] propose the Retro* algorithm, who solve this problem with neural guided A* search. Then Kim et al. [19] future improve this method with self-training and forward models. Meanwhile, the Monte Carlo tree search (MCTS) based methods have also been applied to this task [17, 27, 29, 35]. Despite their achievements, all of these works are based on tree search and inevitably suffer from intra-target duplication. Meanwhile, there are no special designs for inter-target duplication in these methods, and they handle each target separately. The key difference is that our method is a graph-based method without duplicated nodes in trees. Furthermore, our method is optimized for searching a batch of targets together.

5.2 Single-step retrosynthesis

In recent years, many deep learning based single-step retrosynthesis method have been proposed with promising results [6, 8, 10, 28, 30, 33, 36]. Nevertheless, the goal of single-step retrosynthesis is only to predict reactions of one step, not considering the whole retrosynthetic route. This is the crucial difference between single-step retrosynthesis and retrosynthetic planning. The single-step retrosynthesis models are critical component of retrosynthetic planning. Therefore, we use the same single-step model architecture as baseline systems.

5.3 Reinforcement learning

Retrosynthetic planning can be seen as a single player game where the agent (i.e., deep learning models or chemists) manipulates the molecules to a successful state where all ingredients are available. Reinforcement learning based methods have achieved significant results on many conventional games such as GO [31, 32], chess [31], poker [4], and mahjong [24]. However, retrosynthetic planning is different from those conventional games because of the AND-OR relation and inter-target and intra-target duplication. Therefore, we propose to use the AND-OR graph with GNN guidance to handle the search progress.

6 CONCLUSION

This work proposes a novel graph-based search algorithm for the retrosynthetic planning task and a GNN guided search policy. Compared with the conventional tree-based methods, our algorithm can reduce the duplication of molecules in the search process. Furthermore, the GNN policy can better handle the complexity of the AND-OR structure of the search graph and suggest that the node expand more effectively. Therefore, our method is more efficient in single-target planning. Furthermore, our method can naturally search a batch of targets and eliminate inter-target duplication. Our method gets a higher performance over previous systems on two

benchmark datasets. More specifically, we achieved a 99.45% success rate with 500 steps limit in the USPTO dataset, outperforming previous state-of-the-art with 2.6 points.

ACKNOWLEDGMENTS

This work was supported by National Natural Science Foundation of China (NSFC Grant No. 62122089 and No. 61876196), Beijing Outstanding Young Scientist Program NO. BJJWZYJH012019100020098, and Intelligent Social Governance Platform, Major Innovation & Planning Interdisciplinary Platform for the "Double-First Class" Initiative, Renmin University of China. This work was also supported in part by Independent Research Fund Denmark under agreement 8048-00038B. We wish to acknowledge the support provided and contribution made by Public Policy and Decision-making Research Lab of RUC. Rui Yan is supported by Beijing Academy of Artificial Intelligence (BAAI).

REFERENCES

[1] Abien Fred Agarap. 2018. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375* (2018). https://arxiv.org/abs/1803.08375

[2] FR Batchelor, F Po Doyle, JHC Nayler, and GN Rolinson. 1959. Synthesis of penicillin: 6-aminopenicillanic acid in penicillin fermentations. *Nature* 183, 4656 (1959), 257–258.

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

[4] Noam Brown and Tuomas Sandholm. 2019. Superhuman AI for multiplayer poker. *Science* 365, 6456 (2019), 885–890. https://www.science.org/doi/abs/10.1126/science.aay2400

[5] Binghong Chen, Chengtao Li, Hanjun Dai, and Le Song. 2020. Retro*: Learning Retrosynthetic Planning with Neural Guided A* Search. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 1608–1616. http://proceedings.mlr.press/v119/chen20k.html

[6] Shuan Chen and Yousung Jung. 2021. Deep Retrosynthetic Reaction Prediction using Local Reactivity and Global Attention. *JACS Au* (2021), jacsau.1c00246. https://pubs.acs.org/doi/10.1021/jacsau.1c00246

[7] Connor W Coley, William H Green, and Klavs F Jensen. 2019. RDChiral: An RDKit wrapper for handling stereochemistry in retrosynthetic template extraction and application. *JCIM* 59, 6 (2019), 2529–2537.

[8] Hanjun Dai, Chengtao Li, Connor W. Coley, Bo Dai, and Le Song. 2019. Retrosynthesis Prediction with Conditional Graph Logic Network. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 8870–8880. https://proceedings.neurips.cc/paper/2019/hash/0d2b2061826a5df3221116a5085a6052-Abstract.html

[9] Jingxin Dong, Mingyi Zhao, Yuansheng Liu, Yansen Su, and Xiangxiang Zeng. 2021. Deep learning in retrosynthesis planning: datasets, models and tools. *Briefings in Bioinformatics* 00 (2021), 1–15. Issue August.

[10] Yang Fan, Yingce Xia, Jinhua Zhu, Lijun Wu, Shufang Xie, and Tao Qin. 2021. Back translation for molecule generation. *Bioinformatics* (2021). https://doi.org/10.1093/bioinformatics/btab817 btab817.

[11] Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

[12] Hans Finkelstein. 1910. Darstellung organischer Jodide aus den entsprechenden Bromiden und Chloriden. *Berichte der deutschen chemischen Gesellschaft* 43, 2 (1910), 1528–1532.

[13] William Finnigan, Lorna J. Hepworth, Sabine L. Flitsch, and Nicholas J. Turner. 2021. RetroBioCat as a computer-aided synthesis planning tool for biocatalytic reactions and cascades. *Nature Catalysis* 4 (2021), 98–104. Issue 2. https://www.nature.com/articles/s41929-020-00556-z

[14] Peng Han, Peilin Zhao, Chan Lu, Junzhou Huang, Jiaxiang Wu, Shuo Shang, and Bin Yao. 2022. GNN-Retro: Retrosynthetic planning with Graph Neural Networks. In *AAAI*.

[15] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics* 4, 2 (1968), 100–107.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 770–778. https://doi.org/10.1109/CVPR.2016.90

[17] Siqi Hong, Hankz Hankui Zhuo, Kebing Jin, and Zhanwen Zhou. 2021. Retrosynthetic Planning with Experience-Guided Monte Carlo Tree Search. *CoRR* (2021). http://arxiv.org/abs/2112.06028

[18] Joonsoo Jeong, Nagyeong Lee, Yongbeom Shin, and Dongil Shin. 2021. Intelligent generation of optimal synthetic pathways based on knowledge graph inference and retrosynthetic predictions using reaction big data. *Journal of the Taiwan Institute of Chemical Engineers* (2021). https://doi.org/10.1016/j.jtice.2021.07.015

[19] Junsu Kim, Sungsoo Ahn, Hankook Lee, and Jinwoo Shin. 2021. Self-Improved Retrosynthetic Planning. *arXiv:2106.04880 [cs, q-bio]* (2021). https://arxiv.org/abs/2106.04880[cs,q-bio]

[20] Sunghwan Kim, Jie Chen, Tiejun Cheng, Asta Gindulyte, Jia He, Siqian He, Qingliang Li, Benjamin A Shoemaker, Paul A Thiessen, Bo Yu, Leonid Zaslavsky, Jian Zhang, and Evan E Bolton. 2020. PubChem in 2021: new data content and improved web interfaces. *Nucleic Acids Research* 49, D1 (2020), D1388–D1395. https://doi.org/10.1093/nar/gkaa971

[21] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6980

[22] Akihiro Kishimoto, Beat Buesser, Bei Chen, and Adi Botea. 2019. Depth-First Proof-Number Search with Heuristic Edge Cost and Application to Chemical Synthesis Planning. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 7224–7234. https://proceedings.neurips.cc/paper/2019/hash/4fc28b7093b135c21c7183ac07e928a6-Abstract.html

[23] Vladimir I Levenshtein et al. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, Vol. 10. Soviet Union, MAIK Nauka/Interperiodica, address, 707–710.

[24] Junjie Li, Sotetsu Koyamada, Qiwei Ye, Guoqing Liu, Chao Wang, Ruihan Yang, Li Zhao, Tao Qin, Tie-Yan Liu, and Hsiao-Wuen Hon. 2020. Suphx: Mastering Mahjong with Deep Reinforcement Learning. arXiv:2003.13590 [cs.AI]

[25] S. Lloyd. 1982. Least squares quantization in PCM. *IEEE Transactions on Information Theory* 28, 2 (1982), 129–137.

[26] David Rogers and Mathew Hahn. 2010. Extended-connectivity fingerprints. 50, 5 (2010), 742–754.

[27] John S Schreck, Connor W Coley, and Kyle J.M. Bishop. 2019. Learning Retrosynthetic Planning through Simulated Experience. *ACS Central Science* 5 (2019), 970–981. Issue 6. http://pubs.acs.org/journal/acscii

[28] Marwin Segler, Mike Preuß, and Mark P Waller. 2017. Towards" alphachem": Chemical synthesis planning with tree search and deep neural network policies. *arXiv preprint arXiv:1702.00020* (2017). https://arxiv.org/abs/1702.00020

[29] Marwin HS Segler, Mike Preuss, and Mark P Waller. 2018. Planning chemical syntheses with deep neural networks and symbolic AI. *Nature* 555, 7698 (2018), 604–610.

[30] Chence Shi, Minkai Xu, Hongyu Guo, Ming Zhang, and Jian Tang. 2020. A Graph to Graphs Framework for Retrosynthesis Prediction. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 8818–8827. http://proceedings.mlr.press/v119/shi20d.html

[31] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362, 6419 (2018), 1140–1144. https://www.science.org/doi/abs/10.1126/science.aar6404

[32] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *nature* 550, 7676 (2017), 354–359.

[33] Vignesh Ram Somnath, Charlotte Bunne, Connor W Coley, Andreas Krause, and Regina Barzilay. 2020. Learning graph models for template-free retrosynthesis. *arXiv preprint arXiv:2006.07038* (2020). https://arxiv.org/abs/2006.07038

[34] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.* 15, 1 (2014), 1929–1958.

[35] Xiaoxue Wang, Yujie Qian, Hanyu Gao, Connor W Coley, Yiming Mo, Regina Barzilay, and Klavs F Jensen. 2020. Towards efficient discovery of green synthetic pathways with Monte Carlo tree search and reinforcement learning. *Chemical Science* 11 (2020), 10959–10972. Issue 40.

[36] Chaochao Yan, Qianggang Ding, Peilin Zhao, Shuangjia Zheng, JINYU YANG, Yang Yu, and Junzhou Huang. 2020. RetroXpert: Decompose Retrosynthesis Prediction Like A Chemist. *Advances in Neural Information Processing Systems* 33 (2020), 11248–11258. https://proceedings.neurips.cc/paper/2020/file/819f46e52c25763a55cc642422644317-Paper.pdf

A DATA DETAILS

We use two datasets to evaluate our method. The first one is a widely used benchmark USPTO dataset that was introduced by Retro* [5]. The dataset is extracted from the United States Patent Office data that contains about $3.8M$ chemical reactions. Chen et al. [5] processed this data and collected $299,202$ training routes, $65,274$ validation routes, and 190 test routes for retrosynthetic planning task. They select the 190 test routes to guarantee the difficulty of evaluating the algorithm capacity. However, the scale of test data is still small. Therefore, we created an extra test set named USPTO-EXT that contains $10\times$ sizes of test routes of USPTO. To achieve this goal, we first collect $10M$ molecules from PubChem [20] and remove those already in training, validation, or test set of USPTO. Next, we compute the Levenshtein distance [23] between molecules in PubChem and molecules in USPTO. For each target in USPTO test set, we keep the top 10 most similar molecules from PubChem for the USPTO-EXT testset. Therefore, we finally collected $1,900$ test targets in USPTO-EXT. For single-step model training, Chen et al. [5] also provided about $1.3M$ reactions. In addition, we follow [19] to create augmented training data using forward and backward models. We use the same forward (synthetic) model as Kim et al. [19] and train our backward (retrosynthetic) models. We also use the same data filter threshold $\epsilon = 0.8$ as Kim et al. [19].

For GNN training data, we follow the steps of Section 3.3 and collected about $140k$ graphs. We then randomly select $1,000$ graphs for the validation and $1,000$ graphs for the test and use the remaining graphs for training.

Meanwhile, we use the same set of commercially available molecules set $\mathcal{I}$ as [5, 19]. This set is build from *eMolecules* (http://downloads.emolecules.com/free/2019-11-01/) that contains about $231M$ available molecules for our system.

B BASELINE DETAILS

• **Greedy DFS** [17]: This is a classic planning method that always prioritizes the node with max probability. We use tree-based Greedy DFS with 10 as max search depth following previous works.

• **DFPN-E** [22]: This is a method based on Depth-First Proof-Number (DFPN) search.

• **MCTS-rollout** [17]: This is an MCTS method where nodes are molecules and edges are reactions. [17] is an MCTS method where nodes are molecules and edges are reactions.

• **EG-MCTS** and **EG-MCTS-0** [17]: They are MCTS methods with experienced guidance and achieved state-of-the-art results. The difference is that **EG-MCTS-0** does not use a value network.

• **Retro*** and **Retro*-0** [5]: They are tree-based A* search methods where Retro* using a value network but Retro*-0 not.

• **Retro*+** and **Retro*+-0** [19]: They improve the **Retro*** and **Retro*-0** method with self-training.

C REPRODUCIBILITY

Our code is based on the Github repository of Retro* and Retro*+. The policy GNN implementation is using the Pytorch Geometric [11]. The USPTO data can be accessed from the original Retro* Github repository at https://github.com/binghong-ml/retro_star. We use a NVIDIA TESLA P40 GPU for the model training and inference. The search process also uses an Intel Xeon E5-2673 v3 CPU.