Retro*: Learning Retrosynthetic Planning with Neural Guided A* Search

Binghong Chen [1] **Chengtao Li** [2] **Hanjun Dai** [3] **Le Song** [1 4]

Abstract

Retrosynthetic planning is a critical task in organic chemistry which identifies a series of reactions that can lead to the synthesis of a target product. The vast number of possible chemical transformations makes the size of the search space very big, and retrosynthetic planning is challenging even for experienced chemists. However, existing methods either require expensive return estimation by rollout with high variance, or optimize for search speed rather than the quality. In this paper, we propose Retro*, a neural-based A*-like algorithm that finds high-quality synthetic routes efficiently. It maintains the search as an AND-OR tree, and learns a neural search bias with off-policy data. Then guided by this neural network, it performs best-first search efficiently during new planning episodes. Experiments on benchmark USPTO datasets show that, our proposed method outperforms existing state-of-the-art with respect to both the success rate and solution quality, while being more efficient at the same time.

1. Introduction

Retrosynthetic planning is one of the fundamental problems in organic chemistry. Given a target product, the goal of retrosynthesis is to identify a series of reactions that can lead to the synthesis of the product, by searching backwards and iteratively applying chemical transformations to unavailable molecules. As thousands of theoretically-possible transformations can all be applied during each step of reactions, the search space of planning will be huge and makes the problem challenging even for experienced chemists.

The one-step retrosynthesis prediction, which predicts a list of possible direct reactants given product, serves as the foundation for realizing the multistep retrosynthetic planning.

---

[1]College of Computing, Georgia Institute of Technology [2]Galixir [3]Google Research, Brain Team [4]Ant Financial. Correspondence to: Binghong Chen <binghong@gatech.edu>.

Existing methods roughly fall into two categories, either template-based or template-free. Each chemical reaction is associated with a reaction template that encodes how atoms and bonds change during the reaction. Given a target product, template-based methods predict the possible reaction templates, and subsequently apply the predicted reaction templates to target molecule to get corresponding reactants. Existing methods include retrosim (Coley et al., 2017), neuralsym (Segler & Waller, 2017) and GLN (Dai et al., 2019). Though conceptually straightforward, template-based methods need to deal with tens or even hundreds of thousands of possible reaction templates, making the classification task hard. Besides, templates are not always available for chemical reactions. Due to these reasons, people have also been developing template-free methods that could directly predict reactants. Most of existing methods employ seq2seq models like LSTM (Liu et al., 2017) or Transformer (Karpov et al., 2019) from neural machine translation literature.

While one-step methods are continuously being improved, most molecules in real world cannot be synthesized within one step. Possible number of synthesis steps could go up to 60 or even more. Since each molecule could be synthesized by hundreds of different possible reactants, the possible synthesis routes becomes countless for a single product. Such huge space poses challenges for efficient searching and planning, even with advanced one-step approaches.

Besides the huge search space, another challenge is the ambiguity in performance measure and benchmarking. It has been extremely hard to quantitatively analyze the performance of any multi-step retrosynthesis algorithms due to the ambiguous definition of 'good synthesis routes', nor are there any benchmark datasets for analyzing designed algorithms. Most common ways for quantitative analysis is to employ domain experts and let them judge if one synthesis route is better than the other based solely on their experiences, which is both time-consuming and costly.

Due to aforementioned challenges, there are less work proposed in the field of multi-step retrosynthetic planning. Previous works using Monte Carlo Tree Search (MCTS) (Segler et al., 2018; 2017) have achieved superior results over neural- or heuristic-based Breadth First Search (BFS). However, MCTS-based methods has several limitations in this setting:

- Each tree node corresponds to a set of molecules instead

of single molecule. This addtional combinatorial aspect make the representation of tree node, and the estimation of its value even harder. Furthermore, reactions do not explicilty appear as nodes in the tree, which prevents their algorithm from exploiting the structure of subproblems.

- As the algorithm depends on online value estimation, the full rollout from vanilla MCTS may not be efficient for the planning need. Furthermore, the algorithm can not exploit historical data in that many good retrosysthesis plans may have been found previously, and "intuitions" on how to plan efficiently may be learned from these histories.

For quantitative evaluation, they have employed numerous domain experts to conduct A-B tests over methods proposed by their algorithm and other baselines.

In this paper, we present a novel neural-guided tree search method, called Retro*[1], for chemical retrosynthesis planning. In our method,

- We explicitly maintain information about reactions as nodes in an AND-OR tree, where a node with "AND" type corresponds to a reaction, and a node with "OR" type corresponds to a molecule. The tree captures the relations between candidate reactions and reactant molecules, which allows us to exploit structure of subproblems corresponding to a single molecule.
- Based on the AND-OR tree representation, we propose an A*-like planning algorithm which is guided by a neural network learned from past retrosynthesis planning experiences. More specifially, The neural network learns a synthesis cost for each molecule, and it helps the search algorithm to pick the most promising molecule node to expand.

Furthermore, we also propose a method for constructing benchmark synthesis routes data given reactions and chemical building blocks. Based on this, we construct a synthesis route dataset from benchmark reaction dataset USPTO. The route dataset is not only useful for quantitative analysis for predicted synthesis routes, but also work as training data for the neural network components in our method.

Below we summarize our contributions:

- We propose a novel learning-based retrosynthetic planning algorithm to learn from previous planning experience. The proposed algorithm outperforms state-of-the-art methods by a large margin on a realworld benchmark dataset.
- Our algorithm framework can induce a search algorithm that guarantees the optimal solution.
- We propose a method for constructing synthesis route

---

[1]Available at https://github.com/binghong-ml/retro_star

datasets for quantitative analysis of multistep retorsynthetic planning methods.

Our planning algorithm is general in the sense that it can also be applied to other machine learning problems such as theorem proving (Yang & Deng, 2019) and hierarchical task planning (Erol, 1996). A synthetic task planning experiment is included in Appendix D to demonstrate the idea. Most related works have been mentioned in the first two sections. For more related works, please refer to Appendix E.

2. Background

In this section, we first state the problem and its background we are tackling in Section 2.1. Then in Section 2.2 and Section 2.3 we describe how MCTS and proof number search fit in the problem setting.

2.1. Problem Statement

One-step retrosynthesis: Denote the space of all molecule as $\mathcal{M}$. The one-step retrosynthesis takes a target molecule $t \in \mathcal{M}$ as input, and predicts a set of source reactants $\mathcal{S} \subset \mathcal{M}$ that can be used to synthesize t. This is the reverse problem of reaction outcome prediction. In our paper, we assume the existence of such one-step retrosynthesis model (or one-step model for simplicity in the rest of the paper) B,

$$B(\cdot): \quad t \to \{R_i, \mathcal{S}_i, c(R_i)\}_{i=1}^k \qquad (1)$$

which outputs at most k reactions R_i, the corresponding reactant sets $\mathcal{S}_i$ and costs $c(R_i)$. The cost can be the actual price of the reaction R_i, or simply the negative log-likelihood of this reaction under model B. A one-step retrosynthesis model can be learned from a dataset of chemical reactions $\mathcal{D}_{train} = \{\mathcal{S}_i, t_i\}$ [2] which have already been discovered by chemists in the past (Coley et al., 2017; Segler & Waller, 2017; Liu et al., 2017; Dai et al., 2019; Karpov et al., 2019).

Retrosynthesis planning. Given a single target molecule $t \in \mathcal{M}$ and an initial set of molecules $\mathcal{I} \subset \mathcal{M}$, we are interested in synthesizing t via a sequence of chemical reactions using reactants that are from or can be synthesized by $\mathcal{I}$. In this case, $\mathcal{I}$ corresponds to a set of molecules that are commercially available. The goal of retrosynthesis planning is to predict a sequence of reactions with reactants in $\mathcal{I}$ and will ultimately arrive at product t.

Instead of performing forward chaining like reasoning that starts from $\mathcal{I}$, a more efficient and commonly used method is to perform backward chaining that starts from the molecule t, and perform a series of one-step retrosynthesis prediction until all the reactants required are from $\mathcal{I}$. Beyond just finding such a synthesis route, our goal is to find the

---

[2]For simplicity we follow the common practice to ignore the reagents and other chemical reaction conditions.

retrosynthesis plan that are:

- High-quality:
    - The entire retrosynthesis plan should be chemically sound with high probability;
    - The reactants or chemical reactions required should have as low cost as possible;
- Efficient: Due to the synthesis effort, the number of retrosynthesis steps should be limited.

Our proposed Retro* is aiming at finding the best retrosynthesis plan with respect to above criteria in limited time. To achieve this, we also assume that the quality of a solution can be measured by the reaction cost, where such cost is known to our model.
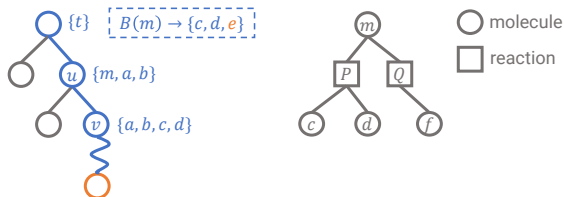
2.2. Monte Carlo Tree Search

Figure 1. **Left**: MCTS (Segler et al., 2018) for retrosynthesis planning. Each node represents a set of molecules. Orange nodes/molecules are available building blocks; **Right**: AND-OR stump illustration of $B(m) = P, Q$. Reaction P requires molecule c and d. Reaction Q requires molecule f. Either P or Q can be used to synthesize m.

The Monte Carlo Tree Search (MCTS) has achieved ground breaking successes in two player games, such as GO (Silver et al., 2016; 2017). Its variant, UCT (Kocsis & Szepesvári, 2006), is especially powerful for balancing exploration and exploitation in online learning setting, and has been employed in Segler et al. (2018) for retrosynthesis planning. Specifically, as illustrated in Figure 1, the tree search start from the target molecule t. Each node u in the current search tree T represents a set of molecules $\mathcal{M}_u$. Each child node $v \in ch(u)$ of u is obtained by selecting one molecule $m \in \mathcal{M}_u$ and a one-step retrosynthesis reaction $(R_{uv}, \mathcal{S}_{uv}, c(R_{uv})) \in B(m)$, where the resulting node v contains molecule set $\mathcal{M}_v = (\mathcal{S}_{uv} \cup \mathcal{M}_u) \setminus \{m\} \setminus \mathcal{I}$.

Despite its good performance, MCTS formulation for retrosynthesis planning has several limitations. First, the roll-out needed in MCTS makes it time-consuming, and unlike in two-player zero-sum games, the retrosynthesis planning is essentially a single player game where the return estimated by random rollouts could be highly inaccurate. Second, since each tree node is a set of molecules instead of a single molecule, the combinatorial nature of this representation brings the sparsity in the variance estimation.

2.3. Proof Number Search and Variants

The proof-number search (PNS) (Allis et al., 1994) is a game tree search that is designed for two-player game with binary goal. It tries to either prove or disprove the root node as fast as possible. In the retrosynthesis planning scenario, this corresponds to either proving the target molecule t by finding a feasible planning path, or concluding that it is not synthesizable.

AND-OR Tree: The search tree of PNS is an AND-OR tree T, where each AND node needs all its children to be proved, while OR node requires at least one to be satisfied. Each node $u \in T$ is associated with a proof number $pn(u)$ that defines the minimum number of leaf nodes to be proved in order to prove u. Similarly, the disproof number $dn(u)$ finds the minimum number of leaf nodes needed to disprove u. With such definition, we can recursively define these numbers for internal nodes. Specifically, for AND node u,

$$pn(u) = \sum_{v \in ch(u)} pn(v), dn(u) = \min_{v \in ch(u)} dn(v)$$

$$\text{and for proved nodes: } pn(u) = 0, dn(u) = +\infty \quad (2)$$

and for OR node u, we have

$$pn(u) = \min_{v \in ch(u)} pn(v), dn(u) = \sum_{v \in ch(u)} dn(v)$$

$$\text{and for disproved node: } pn(u) = +\infty, dn(u) = 0 \quad (3)$$

Represent retrosynthesis planning using AND-OR tree: As illustrated in Figure 1, the application of one-step retrosynthesis model B on molecule m can be represented using one block of AND-OR tree (denoted as AND-OR stump), with molecule node as 'OR' node and reaction node as 'AND' node. This is because a molecule m can be synthesized using **any one** of its children reactions (or-relation), and each reaction node requires **all** of its children molecules (and-relation) to be ready.

The search of PNS starts from the root node every time, and selects the child node with either minimum proof number or minimum disproof number, depends on whether the current node is an OR node or AND node, respectively. The process ends when a leaf node is reached, which can be either reaction or molecule node to be expanded. And after one step of retrosynthesis expansion, all the $pn(\cdot)$ and $dn(\cdot)$ of nodes along the path back to the root will be updated. The two-player game in this sense comes from the interleaving behavior of selecting proof and disproof numbers, where the first 'player' tries to prove the root while the second 'player' tries to disprove it. As both of the players behave optimally when the proof/disproof numbers are accurate, such perspective would bring the efficiency for finding a feasible synthesis path or prove that it is not synthesizable.

Variant: There have been several variants to improve different aspects of PNS, including different traversal strategy, different initialization methods of $pn(\cdot)$ and $dn(\cdot)$ for newly added nodes. The most recent work DFPN-E (Kishimoto

Algorithm 1: Retro*(t)

---
1 Initialize $T = (\mathcal{V}, \mathcal{E})$ with $\mathcal{V} \leftarrow \{t\}$, $\mathcal{E} \leftarrow \emptyset$;
2 **while** *route not found* **do**
3 $m_{next} \leftarrow \text{argmin}_{m \in \mathcal{F}(T)} V_t(m)$;
4 $\{R_i, \mathcal{S}_i, c(R_i)\}_{i=1}^{k} \leftarrow B(m_{next})$;
5   **for** $i \leftarrow 1$ **to** k **do**
6   Add R_i to T under m_{next};
7   **for** $j \leftarrow 1$ **to** $|\mathcal{S}_i|$ **do**
8   Add $\mathcal{S}_{ij}$ to T under R_i;
9   Update $V_t(m)$ for m in $\mathcal{F}(T)$;
10 **return** route;

---

et al., 2019) builds on top of the depth-first variant of PNS with an additive cost in addition to classical update rule in Eq (3). Specifically, for an unsolved OR node,

$$pn(u) = \min_{v \in ch(u)} (h(u,v) + pn(v)) \qquad (4)$$

Here $h(u,v)$ is the function of the cost of corresponding one-step retrosynthesis. Together with manually defined thresholds, this method addresses the *lopsided* problem in retrosynthesis planning, *i.e.*, the imbalance of branching factor between AND and OR nodes.

The variants of PNS has shown some promising results over MCTS for retrosynthesis planning. However, the two-player game formulation is designed for the speed of a proof, not necessarily the overall solution quality. Moreover, existing works rely on human expert to design $pn(\cdot), dn(\cdot)$ and thresholds during search. This makes it not only time-consuming to tune, but also hard to generalize well when solving new target molecule t or dealning with new one-step model or reaction data.

3. Retro* Search Algorithm

Our proposed Retro* is a retrosynthetic planning algorithm that works on the AND-OR search tree. It is significantly different from PNS which is also based on AND-OR tree, or other MCTS based methods in the following ways:

- Retro* utilizes AND-OR tree for *single* player game which only utilizes the global value estimation. This is different from PNS which models the problem as *two-player* game with both proof numbers and disproof numbers. The distinction of the objective makes Retro* advantageous in finding best retrosynthetic routes.
- Retro* estimates the future value of frontier nodes with neural network that can be trained using historical retrosynthesis planning data. This is different from the expensive rollouts used in Segler et al. (2018), or the human designed heuristics in Kishimoto et al. (2019). This not only enables more accurate prediction during expansion, but also generalizes the knowledge learned from existing

planning paths.

3.1. Overview of Retro*

Retro* (Algorithm 1) is a best-first search algorithm, which exploits neural priors to directly optimize for the quality of the solution. The search tree T is an AND-OR tree, with molecule node as 'OR' node and reaction node as 'AND' node. It starts the search tree T with a single root molecule node that is the target molecule t. At each step, it selects a node u in the frontier of T (denoted as $\mathcal{F}(T)$) according to the value function. Then it expands u with the one-step model $B(u)$ and grows T with one AND-OR stump. Finally the nodes with potential dependency on u will be updated. Below we first provide a big picture of the algorithm by explaining these steps one by one, then we look into details of value function design and its update in Section 3.2 and Section 3.3, respectively. Figure 2 summarizes these steps in high level.

Selection: Given a search tree T, we denote the molecule nodes as $\mathcal{V}^m(T)$ and reaction nodes as $\mathcal{V}^r(T)$, where the total nodes in T will be $\mathcal{V}(T) = \mathcal{V}^m(T) \cup \mathcal{V}^r(T)$. The frontier $\mathcal{F}(T) \subseteq \mathcal{V}^m(T)$ contains all the molecule nodes in T that haven't been expanded before. Since we want to minimize the total cost of the final solution, an ideal option to expand next would be the molecule node which is part of the best synthesis plan.

Suppose we already have a value function oracle $V_t(m|T)$ which tells us that under the current search tree T, the cost of the best plan that contains m for synthesizing target t. We can use it to select the next node to expand:

$$m_{next} = \text{argmin}_{m \in \mathcal{F}(T)} V_t(m|T) \qquad (5)$$

A proper design of such $V_t(m|T)$ would not only improve search efficiency, but can also bring theoretical guarantees.

Expansion: After picking the node m with minimum cost estimation $V_t(m|T)$, we will expand the search tree with k one-step retrosynthesis proposals from $B(m)$. Specifically, for each proposed retrosynthesis reaction $(R_i, \mathcal{S}_i, c(R_i)) \in B(m)$, we create a reaction node $R = R_i$ under node m, and for each molecule $m' \in \mathcal{S}_i$, we create a molecule node under the reaction node R. This will create an AND-OR stump under node m. Unlike in MCTS (Segler et al., 2018) where multiple calls to $B(\cdot)$ is needed till a terminal state during rollout, here the expansion only requires a single call to the one-step model.

Update: Denote the search tree T after expansion of node m to be T'. Such expansion obtains the corresponding cost information for one-step retrosynthesis. we utilize this more direct information to update $V_t(\cdot|T')$ of all other relevant nodes to provide a more accurate estimation of total cost.
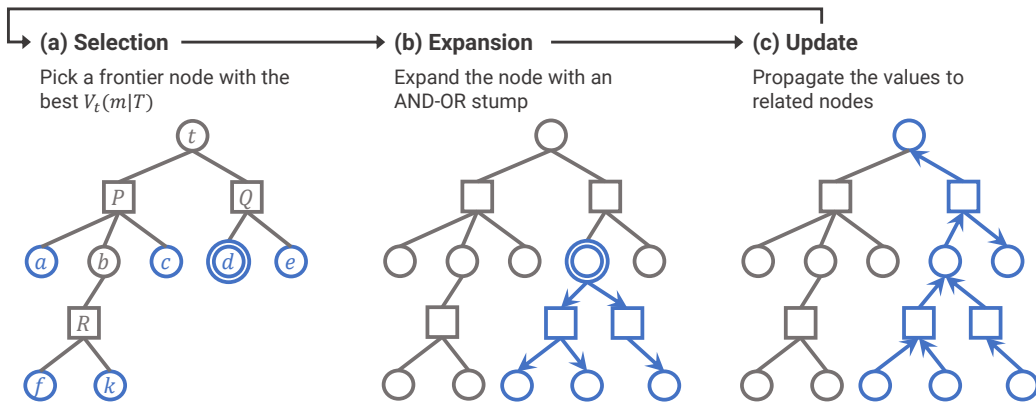
Figure 2. Retro* algorithm framework. We use circles to represent molecule nodes, and squares to represent reaction nodes. An iteration consists of three phases. In the selection phase, one of the frontier molecule nodes is selected according to the cost estimation $V_t(m|T)$. Then the an AND-OR stump is expanded from the selected node. All the new reactions and molecules are added to the tree. Finally the values inside the tree are updated using the V_ms from the newly added molecules. The left-most figure also serves as the illustration for computing $V_t(f|T)$. $V_t(f|T) = g_t(f|T) + h_t(f|T)$, where $g_t(f|T) = c(P) + c(R)$, and $h_t(f|T) = V_a + V_c + V_f + V_k$.

3.2. Design of $V_t(m|T)$

To properly design $V_t(m|T)$, we borrow the idea from A* algorithm. A* algorithm is a best-first search algorithm which uses the cost from start $g(\cdot)$ together with the estimation of future cost $h(\cdot)$ to select move. When such estimation is admissible, it will be guaranteed to return the optimal solution. Inspired by the A* algorithm, we decompose the value function into two parts:

$$V_t(m|T) = g_t(m|T) + h_t(m|T) \quad (6)$$

where $g_t(m|T)$ is the cost of current reactions that have happened in T, if m should be in the final route, and $h_t(m|T)$ is the estimated cost for future reactions needed to complete such planning. Instead of explicitly calculate these two separately, we show an equivalent but simpler way to calculate $V_t(\cdot|T)$ directly.

Specifically, we first define $V_m(m|\emptyset)$, which is a boundary case of the value function oracle V that simply tells how much cost is needed to synthesize molecule m. For the simplicity of notation, we denote it as V_m. Then we define the *reaction number* function $rn(\cdot|T) : \mathcal{V}(T) \mapsto \mathbb{R}$ that is inspired by proof number but with different purpose:

$$rn(R|T) = c(R) + \sum_{m \in ch(R)} rn(m|T)$$

$$rn(m|T) = \begin{cases} V_m, & m \in \mathcal{F}(T) \\ \min_{R \in ch(m)} rn(R|T), & \text{otherwise} \end{cases} \quad (7)$$

where $rn(R|T)$ and $rn(m|T)$ calculate for reaction node and molecule node, respectively. The reaction number tells the minimum estimated cost needed for a molecule or reaction to happen in the current tree. We further define $pr(u|T) : \mathcal{V}(T) \mapsto \mathcal{V}(T)$ to get the parent node of u, and $\mathcal{A}(u|T)$ be all the ancestors of node u. Note that $pr(m|T) \in \mathcal{V}^r(T), \forall m \in \mathcal{V}^m(T)$ and vise versa. Then

function $V_t(m|T)$ will be:

$$\begin{aligned} V_t(m|T) = & \sum_{r \in \mathcal{A}(m|T) \cap \mathcal{V}^r(T)} c(r) \\ & + \sum_{m' \in \mathcal{V}^m(T), pr(m') \in \mathcal{A}(m|T)} rn(m'|T) \quad (8) \end{aligned}$$

The first summation calculates all the reaction cost that has happened along the path from node m to root. Additionally, $\forall R \in \mathcal{A}(m|T) \cap \mathcal{V}^r(T)$, the child node $m' \in ch(R)$ should also be synthesized, as each such reaction node R is an AND node. This requirement is captured in the second summation of Eq (8). We can see that implicitly $g_t(m|T)$ sums up the cost associated with the reaction nodes in this route related to m, and $h_t(m|T)$ takes all the terms related to $V_\cdot$ in Eq (7).

In Figure 2 we demonstrate the calculation of $V_t(m|T)$ with a simple example. Notice that we can compute the parts that relevant to $g_t(\cdot|T)$ with existing information. But we can only estimate the part of $h_t(\cdot|T)$ since the required reactions are not in the search tree yet. We will show how to learn this future estimation in Section 4.2.

3.3. Updating $V_t(m|T)$

After a node m is expanded, there are several components needed to be updated to maintain the search tree state.

Update $rn(\cdot|T)$: Following Eq (7), the reaction number for newly created molecule nodes u under the subtree rooted at m will be V_u, and the reaction nodes $R \in ch(m)$ will have the cost $c(R)$ added to the sum of reaction numbers in children. After that, all the nodes $u \in \mathcal{A}(m|T) \cup \{m\}$ would potentially have the reaction number updated following Eq (7). Thus this process requires the computation complexity to be $O(\text{depth}(T))$. However in our implementation, we can update these nodes in a bottom-up fashion that starts from m, and stop anytime when an ancestor node

value doesn't change. This would speed up the update.

Update $V_t(\cdot|T)$**:** Let $\mathcal{A}'(m|T) \subseteq (\mathcal{A}(m|T) \cup \{m\}) \cap \mathcal{V}^m(T)$ be the set of molecule nodes that have reaction number being updated in the stage above. From Eq (8) we can see, for any molecule node $u \in \mathcal{F}(T)$, $V_t(u|T)$ will be recalculated if $\{m' : pr(m') \in \mathcal{A}(u|T)\} \cap \mathcal{A}'(m|T) \neq \emptyset$.

Remark: The expansion of a node m can potentially affect all other nodes in $\mathcal{F}(T)$ in the worst case. However the expansion of a single molecule node m will only affect another node v in the frontier when it is on the current best synthesis solution that composes $V_t(v|T)$. For the actual implementation, we use efficient caching and lazy propagate mechanism, which will guarantee to only update the $V_t(v|T)$ when it is necessary. The implementation details of both above updates can be found in Appendix A.

3.4. Guarantees on Finding the Optimal Solution

Theorem 1 *Assuming V_m or its lowerbound is known for all encountered molecules m, Algorithm 1 is guaranteed to return an optimal solution, if the halting condition is changed to "the total costs of a found route is no larger than* $\text{argmin}_{m \in \mathcal{F}(T)} V_t(m)$*".*

The proof can be found in Appendix B.

Remark 1: If we define the cost of a reaction to be its negative log-likelihood, then 0 is the lowerbound of V_m for any molecule m. The induced algorithm is guaranteed to find the optimal solution.

Remark 2: In practice, due to the limited time budget, we prefer the algorithm to return once a solution is found.

3.5. Extension: Retro* on Graph Search Space

We have been mainly illustrating the technique on a tree structured space. As the retrosynthesis planning is essentially performed on a directed graph (*i.e.*, certain intermediate molecules may share the same reactants, which may further reduce the actual cost), the above calculation can be extended to the general bipartite graph G with edges connecting $\mathcal{V}^m(G)$ and $\mathcal{V}^r(G)$. Due to the potential existence of loops, the calculation of Eq (7) will be performed using shortest path algorithm instead. As there will be no negative loops, shortest path algorithm will still converge. By viewing the search space as tree rather than graph, we may possibly find sub-optimal solution due to the repetition in state representation. However, as loopy synthesis is rare in real world, we mainly focus on the tree structured search in this paper, and will investigate this extension to bipartite graph space search in future work.

4. Estimating V_m from Planning Solutions

Retro* requires the value function oracle V_m to compute $V_t(\cdot|T)$ for expansion node selection. However in practice

it is impossible to obtain the exact value of V_m for every molecule m. Therefore we try to estimate it from previous planning data.

4.1. Representation of V_m

To parameterize V_m for any molecule m, we first compute its Morgan fingerprint (Rogers & Hahn, 2010) of radius 2 with 2048 bits, and feed it into a single-layer fully connected neural network of hidden dimension 128, which then outputs a scalar representing V_m.

4.2. Offline Learning of V_m

Previous work has either used random rollout or human designed heuristics for estimating V_m, which may not be accurate enough to guide the search. Instead of learning it online during planning (Silver et al., 2017), we utilize the existing reactions in the training set $\mathcal{D}_{train}$ to train it.

Specifically, we construct retrosynthesis routes for feasible molecules in $\mathcal{D}_{train}$, where the available set of molecule $\mathcal{M}$ is also given beforehand. The specific construction strategy will be covered in Section 5.1.2. The resulting dataset will be $\mathcal{R}_{train} = \{rt_i = (m_i, v_i, R_i, B(m_i))\}$, where each tuple rt_i contains the target molecule m_i, the best entire route cost v_i, the one-step retrosynthesis candidates $B(m_i)$ which also contains the true one-step retrosynthesis R_i used in the planning solution.

The learning of V_m consists of two parts, namely the value fitting which is a regression loss $\mathcal{L}_{reg}(rt_i) = (V_{m_i} - v_i)^2$ and the consistency learning which maintains the partial order relationship between best one-step solution R_i and other solutions $(R_j, \mathcal{S}_j, c(R_j)) \in B(m_i)$:

$$\mathcal{L}_{con}(rt_i, R_j) = \max\left\{0, v_i + \epsilon - c(R_j) - \sum_{m' \in \mathcal{S}_j} V_{m'}\right\} \tag{9}$$

where ϵ is a positive constant margin to ensure r_i has higher priority for expansion than its alternatives even if the value estimates have tolerable noise. The overall objective is:

$$\min_{V_{(\cdot)}} \quad \mathbb{E}_{rt_i \sim \mathcal{R}_{train}}\Big[\mathcal{L}_{reg}(rt_i) + \\ \lambda \mathbb{E}_{R_j \sim B(m_i) \setminus \{R_i\}} [\mathcal{L}_{con}(rt_i, R_j)]\Big] \tag{10}$$

where λ balances these two losses. In experiment we set it to be 1 by default.

5. Experiments

5.1. Creating Benchmark Dataset

5.1.1. USPTO REACTION DATASET

We use the publicly available reaction dataset extracted from United States Patent Office (USPTO) to train one-step model

Algorithm	Retro*	Retro*-0	DFPN-E+	DFPN-E	MCTS+	MCTS	Greedy DFS
Success rate	86.84%	79.47%	53.68%	55.26%	35.79%	33.68%	22.63%
Time	156.58	208.58	289.42	279.67	365.21	370.51	388.15
Shorter routes	50	52	59	59	18	14	11
Better routes	112	102	22	25	46	41	26

Table 1. Performance summary. Time is measured by the number of one-step model calls, with a hard limit of 500. The number of shorter and better routes are obtained from the comparison against the expert routes, in terms of number of reactions and the total costs.

and extract synthesis routes. The whole dataset consists of $\sim 3.8M$ chemical reactions published up to September 2016. For reactions with multiple products, we duplicate them into multiple ones with one product each. After removing the duplications and reactions with wrong atom mappings, we further extract reaction templates with RD-Chiral [3] for all reactions and discard those whose reactants cannot be obtained by applying reaction templates to their products. The remaining $\sim 1.3M$ reactions are further split randomly into train/val/test sets following $80\%/10\%/10\%$ proportions.

With reaction data, we train a template-based MLP model (Segler & Waller, 2017) for one-step retrosynthesis. Following literature, we formulate the one-step retrosynthesis as a multi-class classification problem, where given a molecule as product, the goal is to predict possible reaction templates. Reactants are obtained by applying the predicted templates to product molecule. There are in total $\sim 380K$ distinct templates. Throughout all experiments, we take the top-50 templates predicted by MLP model and apply them on each product to get corresponding reactant lists.

5.1.2. EXTRACTING SYNTHESIS ROUTES

To train our value function and quantitatively analyze the predicted routes, we construct synthesis routes based on USPTO reaction dataset and a list of commercially available building blocks from *eMolecules* [4]. *eMolecules* consists of $231M$ commercially available molecules that could work as ending points for our searching algorithm.

Given the list of building blocks, we take each molecule that have appeared in USPTO reaction data and analyze if it can be synthesized by *existing* reactions within USPTO training data. For each synthesizable molecule, we choose the shortest-possible synthesis routes with ending points being available building blocks in *eMolecules*.

We obtain validation and test route datasets with slightly different process. For validation dataset, we first combine train and validation *reaction* dataset, and then repeat aforementioned extraction procedure on the combined dataset. Since we extract routes with more reactions, synthesizable

molecules will include those who could not be synthesized with original reactions and those who have shorter routes. We exclude molecules with routes of same length as in training data, and pack the remaining as validation route dataset. We apply similar procedure to test data but make sure that there is no overlap between test and training/validation set.

We further clean the test route dataset by only keeping the routes whose reactions are all covered by the top-50 predictions by the one-step model. To make the test set more challenging, we filter out the easier molecules by running a heuristic-based BFS planning algorithm, and discarding the solved molecules in a fixed time limit. After processing, we obtain 299202 training routes, 65274 validation routes, 189 test routes and the corresponding target molecules.

5.2. Results

We compare Retro* against DFPN-E (Kishimoto et al., 2019), MCTS (Segler et al., 2018) and greedy Depth First Search (DFS) on product molecules in test route dataset described in Section 5.1.2. Greedy DFS always prioritizes the reaction with the highest likelihood. MCTS is implemented with PUCT, where we used the reaction probability provided by the one-step model as the prior to bias the search.

We measure both route quality and planning efficiency to evaluate the algorithm. To measure the quality of a solution route, we compare its total cost as well as its length, *i.e.* number of reactions in the route. The cost function is defined as the negative log-likelihood of the reaction. Therefore, minimizing the total costs is equivalent to maximizing the likelihood of the route. To measure planning effiency, we use the number of calls to the one-step model ($\approx 0.3s$ per call) as a surrogate of time (since it will occupy $> 99\%$ of running time) and compare the success rate under the same time limit.

Performance summary: The performances of all algorithms are summarized in Table 1. Under the time limit of 500 one-step calls, Retro* solves 31% more test molecules than the second best method, DFPN-E. Among all the solutions given by Retro*, 50 of them are shorter than expert routes, and 112 of them are better in terms of the total costs. We also conduct an ablation study to understand the importance of the learning component in Retro* by evaluating its non-learning version Retro*-0. Retro*-0 is obtained from
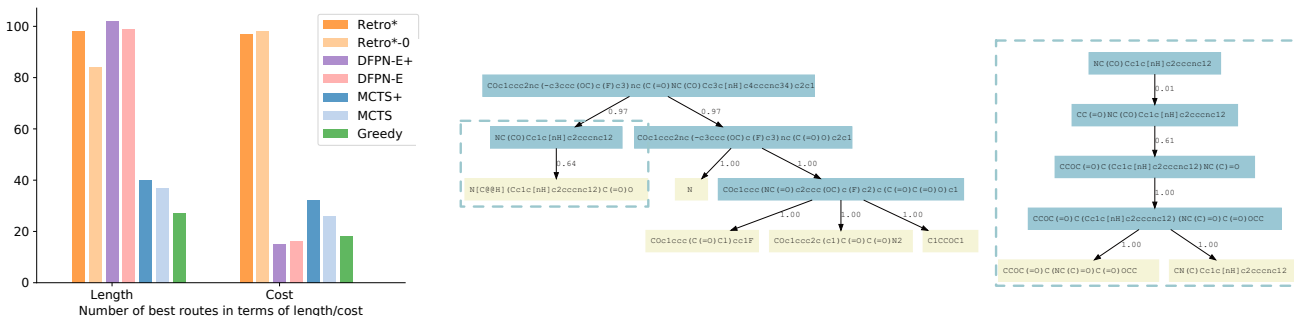
---

[3] https://github.com/connorcoley/rdchiral
[4] http://downloads.emolecules.com/free/2019-11-01/

Figure 3. **Left**: Counts of the best solutions among all algorithms in terms of length/cost; **Mid**: Sample solution route from Retro*. Numbers on the edges are the likelihoods of the reactions. Yellow nodes are building blocks; **Right**: The corresponding dotted box part in the expert route, much longer and less probable than the solution.

Retro* by setting V_m to 0, which is a lowerbound of any valid values. Comparing to baseline methods, Retro*-0 is also showing promising results. However, it is outperformed by Retro* by 6% in terms of success rate, demonstrating the performance gain brought by learning from previous planning experience.

To find out whether MCTS and DFPN-E can benefit from the learned value function oracle V_m in Retro*, we replace the reward estimation by rollout in MCTS and the proof number initialization in DFPN-E by the same V_m, calling the strengthened algorithms MCTS+ and DFPN-E+. Value function helps MCTS as expected due to having a value estimate with less variance than rollout. The performance of DFPN-E is not improved because we dont have a good initialization of the disproof number.
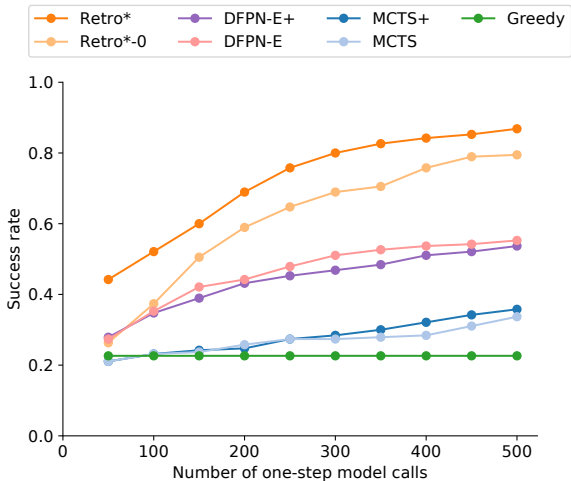

Figure 4. Influence of time limit on performance.

Influence of time limit: To show the influence of time limit on performance, we plot the success rate against the number of one-step model calls in Figure 4. We can see that Retro* not only outperforms baseline algorithms by a large margin at the beginning, but also is improving faster than the baselines, enlarging the performance gap as the time

limit increases.

Solution quality: To evaluate the overall solution quality, for each test molecule, we collect solutions from all algorithms, and compare the route lengths and costs (see Figure 3-left). We only keep the best routes (could be multiple) for each test molecule, and count the number of best routes in total for each method. We find that in terms of total costs, Retro* produces 4× more best routes than the second best method. Even for the length metric, which is not the objective Retro* is optmizing for, it still achieves about the same performance as the best method.

As a demonstration for Retro*'s ability to find high-quality routes, we illustrate a sample solution in Figure 3-mid, where each node represents a molecule. The target molecule corresponds to the root node, and the building blocks are in yellow. The numbers on the edges indicates the likelihoods of successfully producing the corresponding reactions in realworld. The expert route provided shares the exactly the same first reaction and the same right branch with the route found by our algorithm. However, the left branch (Figure 3-right) is much longer and less probable than the corresponding part of the solution route, as shown in the dotted box region in Figure 3-mid. Please refer to Appendix C for more sample solution routes and search tree visualizations.

6. Conclusion

In this work, we propose Retro*, a learning-based retrosynthetic planning algorithm for efficiently finding high-quality routes. Retro* is able to utilize previous planning experience to bias the search on unseen molecules towards promising directions. We also propose a systematic approach for creating a retrosynthesis dataset from publicly available reaction datasets and novel metrics for evaluating solution routes without involving human experts. Experiments on realworld benchmark dataset demonstrate our algorithm's significant improvement over existing methods on both planning efficiency and solution quality.

References

Allis, L. V., van der Meulen, M., and Van Den Herik, H. J. Proof-number search. *Artificial Intelligence*, 66(1):91–124, 1994.

Chen, B., Dai, B., Lin, Q., Ye, G., Liu, H., and Song, L. Learning to plan in high dimensions via neural exploration-exploitation trees. In *International Conference on Learning Representations*, 2020.

Coley, C. W., Rogers, L., Green, W. H., and Jensen, K. F. Computer-assisted retrosynthesis based on molecular similarity. *ACS Central Science*, 3(12):1237–1245, 2017.

Dai, H., Li, C., Coley, C., Dai, B., and Song, L. Retrosynthesis prediction with conditional graph logic network. In *Advances in Neural Information Processing Systems*, pp. 8870–8880, 2019.

Erol, K. *Hierarchical task network planning: formalization, analysis, and implementation*. PhD thesis, 1996.

Guez, A., Weber, T., Antonoglou, I., Simonyan, K., Vinyals, O., Wierstra, D., Munos, R., and Silver, D. Learning to search with MCTSnets. *arXiv preprint arXiv:1802.04697*, 2018.

Hart, P. E., Nilsson, N. J., and Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

Karpov, P., Godin, G., and Tetko, I. A transformer model for retrosynthesis. 2019.

Kishimoto, A., Buesser, B., Chen, B., and Botea, A. Depth-first proof-number search with heuristic edge cost and application to chemical synthesis planning. In *Advances in Neural Information Processing Systems*, pp. 7224–7234, 2019.

Kocsis, L. and Szepesvári, C. Bandit based Monte-Carlo planning. In *European conference on machine learning*, pp. 282–293. Springer, 2006.

Liu, B., Ramsundar, B., Kawthekar, P., Shi, J., Gomes, J., Luu Nguyen, Q., Ho, S., Sloane, J., Wender, P., and Pande, V. Retrosynthetic reaction prediction using neural sequence-to-sequence models. *ACS Central Science*, 3 (10):1103–1113, 2017.

Rogers, D. and Hahn, M. Extended-connectivity fingerprints. *Journal of chemical information and modeling*, 50 (5):742–754, 2010.

Schreck, J. S., Coley, C. W., and Bishop, K. J. Learning retrosynthetic planning through simulated experience. *ACS Central Science*.

Segler, M., Preuß, M., and Waller, M. P. Towards" alphachem": Chemical synthesis planning with tree search and deep neural network policies. *arXiv preprint arXiv:1702.00020*, 2017.

Segler, M. H. and Waller, M. P. Neural-symbolic machine learning for retrosynthesis and reaction prediction. *Chemistry–A European Journal*, 23(25):5966–5971, 2017.

Segler, M. H., Preuss, M., and Waller, M. P. Planning chemical syntheses with deep neural networks and symbolic ai. *Nature*, 555(7698):604, 2018.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of GO with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. Mastering the game of GO without human knowledge. *Nature*, 550(7676):354–359, 2017.

Yang, K. and Deng, J. Learning to prove theorems via interacting with proof assistants. *arXiv preprint arXiv:1905.09381*, 2019.
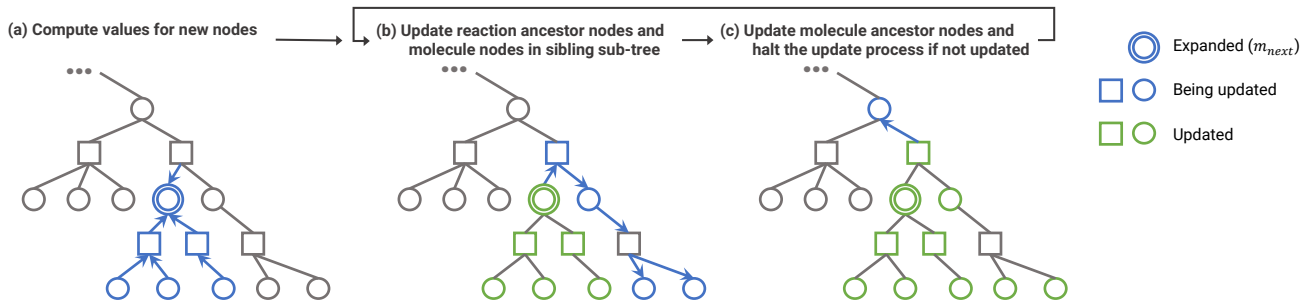
A. Implementation details

Figure 5. Illustration for the update process. Three phases correspond to line 1-8, line 11-16, and line 17-21 in Algorithm 2.

In this section we describe the algorithm details in the update phase of Retro*. The goal of the update phase is to compute the up-to-date $V_t(m|T)$ for every molecule node $m \in \mathcal{F}(T)$. To implement efficient update, we need to cache $V_t(m|T)$ for all $m \in \mathcal{V}^m(T)$. Note that from Eq (8), we can observe the fact that sibling molecule nodes have the same $V_t(m|T)$, *i.e.* $V_t(m_a|T) = V_t(m_b|T)$ if $pr(m_a|T) = pr(m_b|T)$. Therefore instead of storing the value of $V_t(m|T)$ in every molecule node m, we store the value in their common parent via defining $V_t(R|T) = V_t(m|T)$ if $R = pr(m|T)$ for every reaction node $R \in \mathcal{V}^r(T)$.

In our implementation, we cache $V_t(R|T)$ for all reaction nodes $R \in \mathcal{V}^r(T)$ and cache $rn(v|T)$ for all nodes $v \in \mathcal{V}(T)$. Caching values in this way would allow us to visit each related node only once for minimal update.

Algorithm 2: $\texttt{Update}(m_{next}, \{R_i, \mathcal{S}_i, c(R_i)\}_{i=1}^k)$ [5]

---

1 **for** $i \leftarrow 1$ **to** k **do**
2   **for** $m \in \mathcal{S}_i$ **do**
3          $rn(m) \leftarrow V_m$;
4      $rn(R_i) \leftarrow c(R_i) + \sum_{m \in \mathcal{S}_i} rn(m)$;
5      $V_t(R_i) \leftarrow V_t(pr(m_{next})) - rn(m_{next}) + rn(R_i)$;
6 $new\_rn \leftarrow \min_{i \in \{1,2,\cdots,k\}} rn(R_i)$;
7 $delta \leftarrow new\_rn - rn(m_{next})$;
8 $rn(m_{next}) \leftarrow new\_rn$;
9 $m_{current} \leftarrow m_{next}$;
10 **while** $delta \neq 0$ *and* $m_{current}$ *is not root* **do**
11      $R_{current} \leftarrow pr(m_{current})$;
12      $rn(R_{current}) \leftarrow rn(R_{current}) + delta$;
13      $V_t(R_{current}) \leftarrow V_t(R_{current}) + delta$;
14   **for** $m \in ch(R_{current})$ **do**
15   **if** m *is not* $m_{current}$ **then**
16              $\texttt{UpdateSibling}(m, delta)$;
17      $m_{current} \leftarrow pr(R_{current})$;
18      $delta = 0$;
19   **if** $rn(R_{current}) < rn(m_{current})$ **then**
20          $delta \leftarrow rn(R_{current}) - rn(m_{current})$;
21          $rn(m_{current}) \leftarrow rn(R_{current})$;

The update function is summarized in Algorithm 2 and illustrated in Figure 5, which takes in the expanded node m_{next} and the expansion result $\{R_i, \mathcal{S}_i, c(R_i)\}_{i=1}^k$, and performs updates to affected nodes. We first compute the values for new

---

[5]For clarity, we omit the condition on T in the notations.

reactions according to Eq (7) and (8) in line 1-8. Then we update the ancestor nodes of m_{next} in a bottom-up fashion in line 9-21. We also update the molecule nodes in the sibling sub-trees in line 16 and Algorithm 3.

Algorithm 3: UpdateSibling($m, delta$)

---
1   $rn(m|T) \leftarrow rn(m|T) + delta$;
2   **for** $R \in ch(m|T)$ **do**
3   **for** $m' \in ch(R|T)$ **do**
4   UpdateSibling($m', delta$);

Our implementation visits a node only when necessary. When updating along the ancestor path, it immediately stops when the influence of the expansion vanishes (line 10). When updating a single node, we use a $O(1)$ delta update by leveraging the relations derived from Eq (7) and (8), avoiding a direct computation which may require $O(k)$ or $O(\text{depth}(T))$ summations.

B. Guarantees on finding the optimal solution

Since Retro* is a variant of the A* algorithm, we can leverage existing results to prove the theoretical guarantees for Retro*. In this section, we first state the assumptions we make, and then prove the admissibility (Theorem 1) of Retro*.

The theoretical results in this paper build upon the assumption that we can access $\hat{V}_m$, which is a lowerbound for V_m for all molecules m. Note that this is a weak assumption, since we know 0 is a universal lowerbound for V_m.

As we describe in Eq (6), $V_t(m|T)$ can be decomposed into $g_t(m|T)$ and $h_t(m|T)$, where $g_t(m|T)$ is the exact cost of the partial route through m which is already in the tree, and $h_t(m|T)$ is the future costs for frontier nodes in the route which is a summation of a series of V_ms. In practice we use $\hat{V}_m$ in the summation, and arrive at $\hat{h}_t(m|T)$, which is a lowerbound of $h_t(m|T)$, *i.e.* the following lemma.

Lemma 2 *Assuming V_m or its lowerbound is known for all encountered molecules m, then the approximated future costs $\hat{h}_t(m|T)$ in Retro* is a lowerbound of true $h_t(m|T)$.*

We re-state the admissibility result (Theorem 1) in the main text and prove it with existing results in A* literature.

Theorem 1 *(**Admissibility**) Assuming V_m or its lowerbound is known for all encountered molecules m, Algorithm 1 is guaranteed to return an optimal solution, if the halting condition is changed to "the total costs of a found route is no larger than $\operatorname{argmin}_{m \in \mathcal{F}(T)} V_t(m)$".*

Proof Combine Lemma 2 and Theorem 1 in the original A* paper (Hart et al., 1968). ∎

C. Sample search trees and solution routes

In this section, we present two examples of the solution routes and the corresponding search trees for target molecule A and B produced by Retro*.

Solution route for target molecule A/B is illustrated in the top/bottom sub-figure of Figure 6, where a set of edges pointing from the same product molecule to reactant molecules represents an one-step chemical reaction. Molecules on the leaf nodes are all available.

The search trees for molecule A and B are illustrated in Figure 7 and Figure 8. We use reactangular boxes to represent molecules. Yellow/grey/blue boxes indicate available/unexpanded/solved molecules. Reactangular arrows are used to represent reactions. The numbers on the edges pointing from a molecule to a reaction are the probabilities produced by the one-step model. Due to space limit, we only present the minimal tree which leads to a solution.
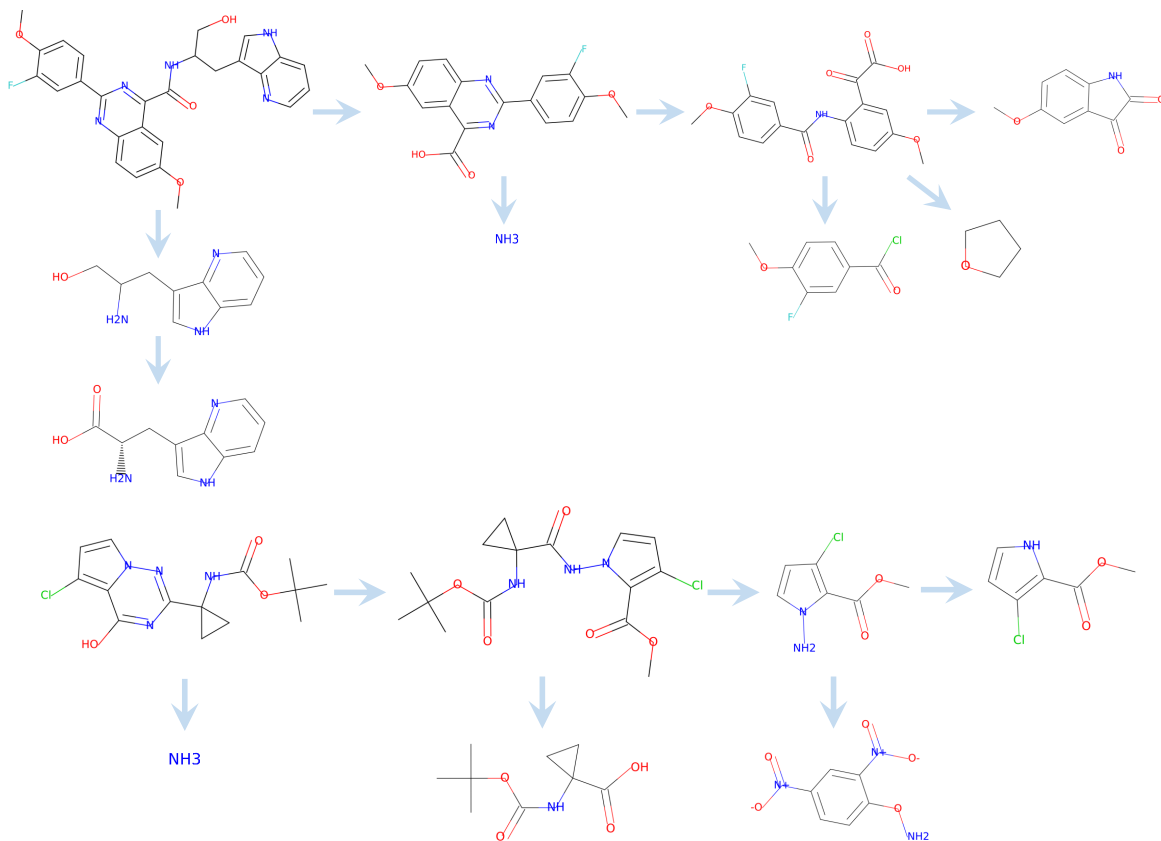
Figure 6. Top/bottom: solution route produced by Retro* for molecule A/B. Edges point from the same product molecule to the reactant molecules represent an one-step chemical reaction.

D. Retro* for hierarchical task planning

As a general planning algorithm, Retro* can be applied to other machine learning problems as well, including theorem proving (Yang & Deng, 2019) and hierarchical task planning (Erol, 1996) (or HTP), etc. Below, we conduct a synthetic experiment on HTP to demonstrate the idea. In the experiment, we are trying to search for a plan to complete a target task. The tasks (OR nodes) can be completed with different methods, and each method (AND nodes) requires a sequence of subtasks to be completed. Furthermore, each method is associated with a nonnegative cost. The goal is to find a plan with minimum total cost to realize the target task by decomposing it recursively until all the leaf task nodes represent primitive tasks that we know how to execute directly. As an example, to travel from home in city A to hotel in city B, we can take either `flight`, `train` or `ship`, each with its own cost. For each method, we have subtasks such as home $\rightarrow$ airport A, `flight`$(A \rightarrow B)$, and airport $B \rightarrow$ hotel. These subtasks can be further realized by several methods.

As usual, we want to find a plan with small cost in limited time which is measured by the number of expansions of task nodes. We use the optimal halting condition as stated in theorem 1. We compare our algorithms against DFPN-E, the best performing baseline. The results are summarized in Table 2 and 3.

Time Limit	15	20	25	30	35
Retro*	.67	.91	.96	.98	1.
Retro*-0	.50	.86	.95	.98	.99
DFPN-E	.02	.33	.74	.93	.97

Table 2. Success rate (higher is better) vs time limit.

As we can see, in terms of success rate, Retro* is slightly better than Retro*-0, and both of them are significantly better than DFPN-E. In terms of solution quality, we compute the approximation ratio (= solution cost / ground truth best solution cost)

Alg	Retro*	Retro*-0	DFPN-E
Avg. AR	1	1	1.5
Max. AR	1	1	3.9

Table 3. AR = Approximation ratio (lower is better), time limit=35.

for every solution, and verify the theoretical guarantee in theorem 1 on finding the best solution.

E. Related Works

Reinforcement learning algorithms (without planning) have also been considered for the retrosynthesis problem. Schreck et al. leverages self-play experience to fit a value function and uses policy iteration for learning an expansion policy. It is possible to combine it with a planning algorithm to achieve better performance in practice.

Learning to search from previous planning experiences has been well studied and applied to Go (Silver et al., 2016; 2017), Sokoban (Guez et al., 2018) and path planning (Chen et al., 2020). Existing methods cannot be directly applied to the retrosynthesis problem since the search space is more complicated, and the traditional representation where a node corresponds to a state is highly inefficient, as we mentioned in the discussion on MCTS in previous sections.
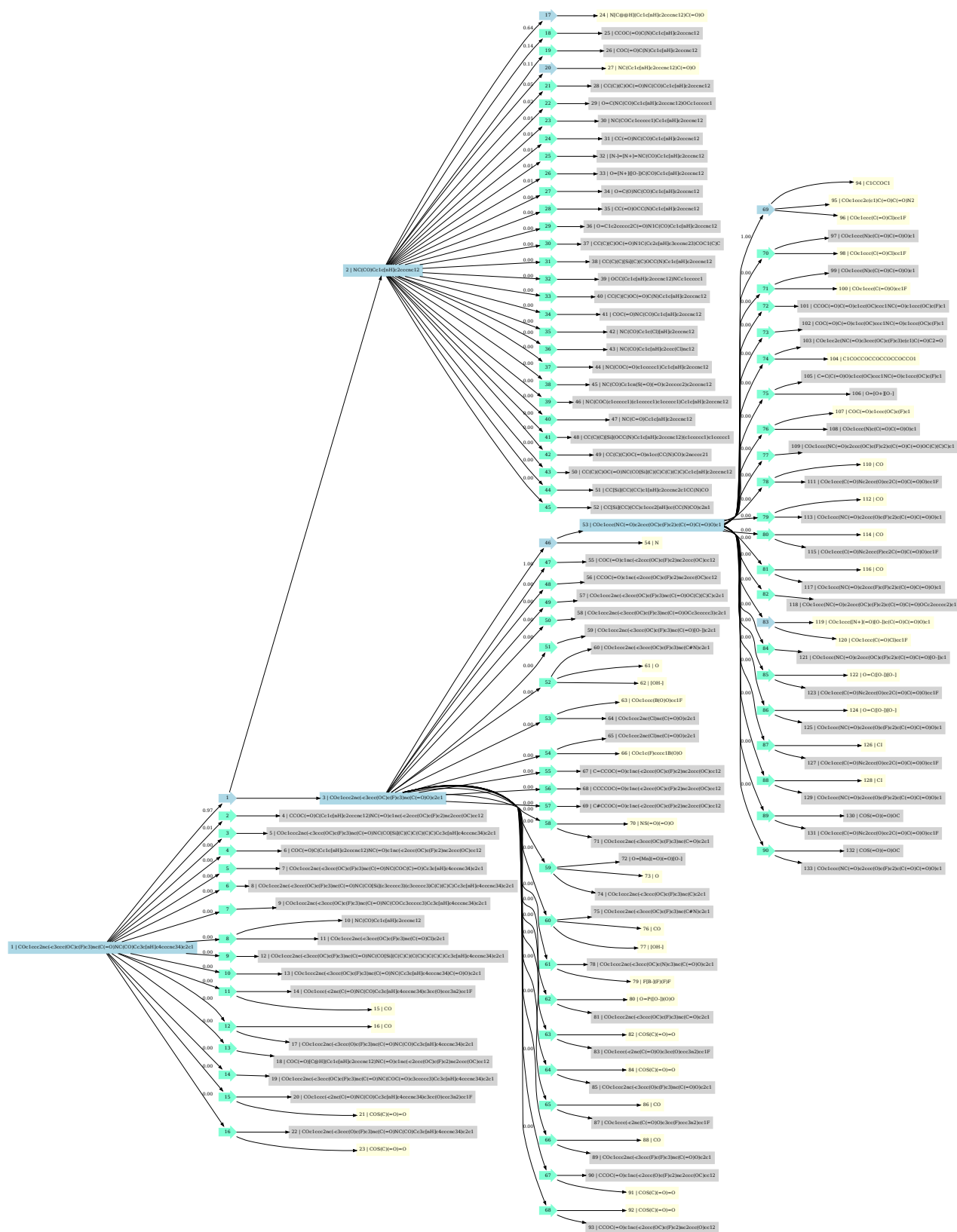
Figure 7. Search tree produced by Retro* for molecule *A*. Reactangular boxes/arrows represent molecules/reactions. Yellow/grey/blue indicate available/unexpanded/solved molecules. Numbers on the edges are the probabilities produced by the one-step model.
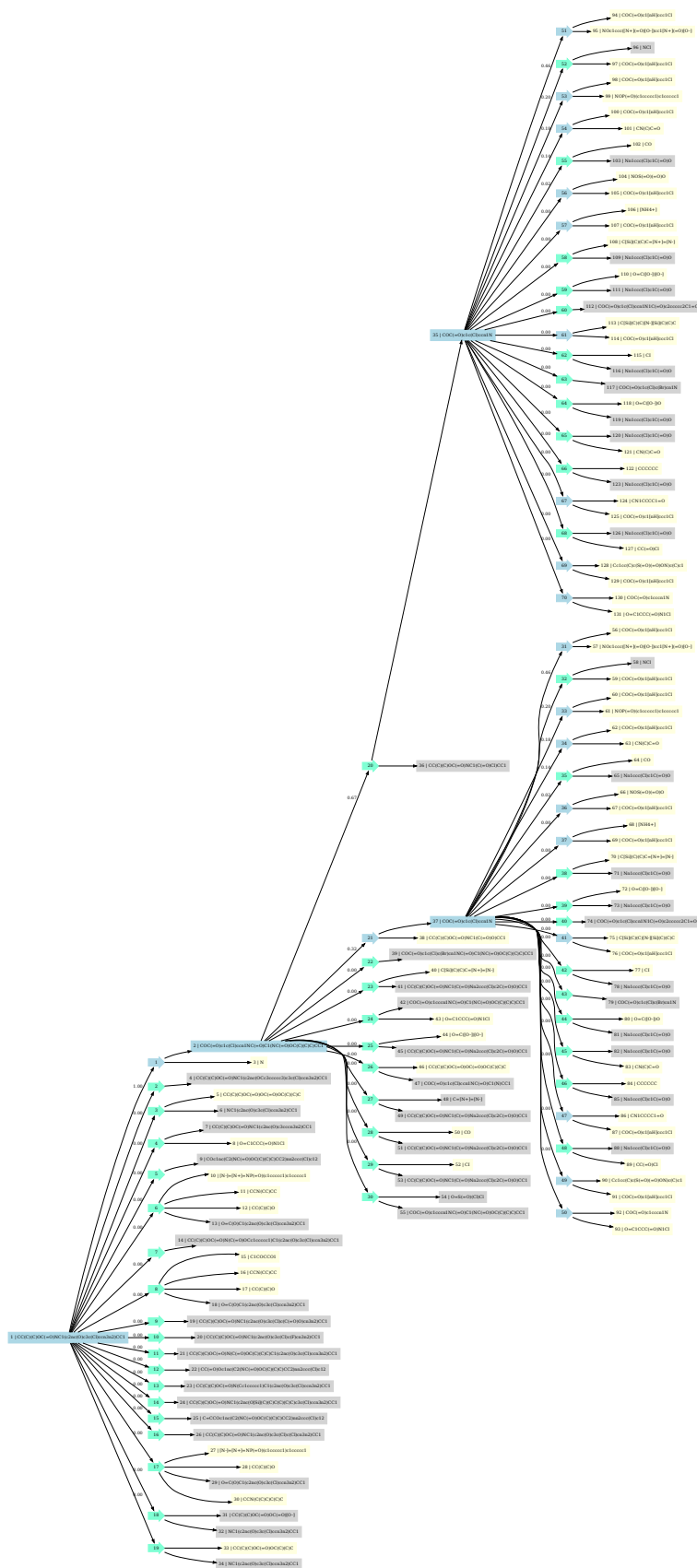
Figure 8. Search tree produced by Retro* for molecule B. Reactangular boxes/arrows represent molecules/reactions. Yellow/grey/blue indicate available/unexpanded/solved molecules. Numbers on the edges are the probabilities produced by the one-step model.