

---

# Learning Retrosynthetic Planning with Chemical Reasoning

---

Anonymous Authors<sup>1</sup>

## Abstract

Retrosynthetic planning is a critical task in chemistry that identifies a series of reactions which lead to the synthesis of a target product. It is challenging even for experienced chemists due to the huge search space brought by the vast number of possible chemical transformations. Existing methods have various limitations, e.g., requiring expensive return estimation by rollout with high variance or optimizing for search speed rather than the quality. In this paper, we propose a retrosynthetic planning framework, `Retro*`, by combining learned neural search biases with chemical reasoning. `Retro*` is a neural-based A\*-like algorithm that finds high-quality synthetic routes efficiently using an AND-OR tree. While it could be directly applied to small molecules, we further extend the framework to handle polymer synthesis. Experiments on benchmark 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 which 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 [5, 17, 6, 13, 10], which predicts a list of possible direct reactants given product, serves as the foundation for realizing the multistep retrosynthetic planning. 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. Previous works on retrosynthetic planning using Monte Carlo Tree Search (MCTS) [18, 16] and Heuristic Depth-First Proof-Number Search (DFPN-E) [11] have achieved superior results over neural- or heuristic-based Breadth First Search (BFS). However, their methods suffer from the poor representation of the search space, expensive rollout estimation, and inconvenient hand-designed heuristics.

In this paper, we present `Retro*` and its extension `PolyRetro`, the learning-based retrosynthetic planning algorithms for small molecules and polymers. Our methods leverage one-step model for chemical reasoning, and learn a neural search bias for guiding synthesis route search. Below we summarize our contributions:

- We propose a novel learning-based retrosynthetic planning algorithm for small molecules to learn from previous planning experience. The proposed algorithm outperforms state-of-the-art methods by a large margin on a realworld benchmark dataset.
- The extended method on polymer is able to recover 53% of ground truth monomers for a real-world polymer dataset using limited training data, significantly outperforming all existing algorithms.

## 2. Background

In this section, we state the retrosynthesis problem and its background we are tackling. The description on how MCTS and proof number search fit in the problem setting are deferred to Appendix A.2 and A.3.

**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  $t$ . 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)  $B$ ,

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

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  $\mathcal{D}_{train} = \{\mathcal{S}_i, t_i\}$  [5, 17, 13, 6, 10].

**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  $\mathcal{I}$ . Retrosynthetic planning algorithms start from the molecule  $t$ , and perform a series of one-step chemical reasoning (retrosynthesis prediction) until all the reactants required are from  $\mathcal{I}$ . Beyond just finding such a synthesis route, our goal is to find the retrosynthesis plan that are:

- High-quality: 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 `Retros*` is aiming at finding the best retrosynthesis plan with respect to above criteria. 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.

### 3. `Retros*` Search Algorithm

#### 3.1. AND-OR tree representation

As illustrated in Figure 5, 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.

#### 3.2. Overview of `Retros*`

`Retros*` (Algorithm 1 in Appendix B) 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 3.3 and Appendix B.2. Figure 1 summarizes these steps in high level. The framework is able to induce an algorithm with theoretical guarantees in Appendix D.

**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:

$$m_{next} = \operatorname{argmin}_{m \in \mathcal{F}(T)} V_t(m|T) \quad (2)$$

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 [18] where multiple calls to  $B(\cdot)$  is needed till a terminal state during rollout, here the expansion only requires a single call to the 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.

#### 3.3. 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:

$$V_t(m|T) = g_t(m|T) + h_t(m|T) \quad (3)$$

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 them 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:

$$\begin{aligned} rn(R|T) &= c(R) + \sum_{m \in \text{ch}(R)} rn(m|T) \\ rn(m|T) &= \begin{cases} V_m, & m \in \mathcal{F}(T) \\ \min_{R \in \text{ch}(m)} rn(R|T), & \text{otherwise} \end{cases} \quad (4) \end{aligned}$$

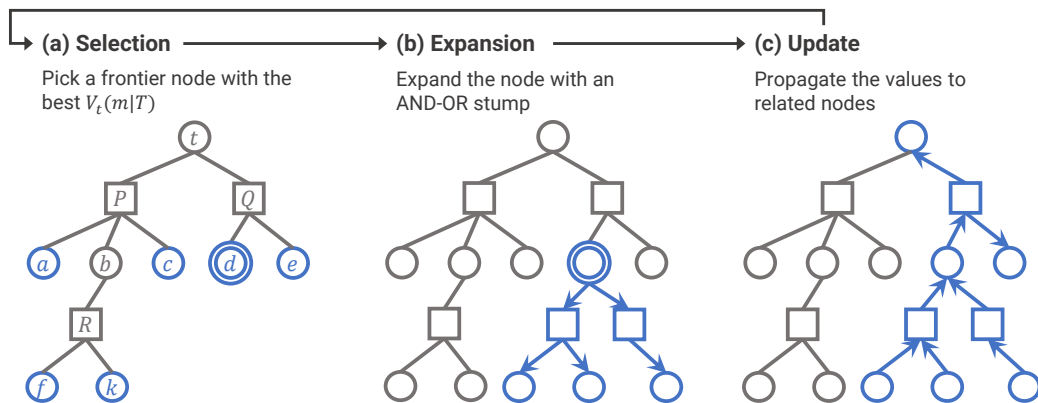


Figure 1. **RetRO\*** algorithm framework. We use circles to represent molecule nodes, and squares to represent reaction nodes. The left-most figure is an example:  $V_t(f|T) = g_t(f|T) + h_t(f|T)$ , where  $g_t(f|T) = c(P) + c(R)$ , and  $h_t(f|T) = V_a + V_c + V_f + V_k$ .

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 vice versa. Then function  $V_t(m|T)$  will be:

$$V_t(m|T) = \sum_{r \in \mathcal{A}(m|T) \cap \mathcal{V}^r(T)} c(r) + \sum_{m' \in \mathcal{V}^m(T), pr(m') \in \mathcal{A}(m|T)} rn(m'|T) \quad (5)$$

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 (5). 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  $V$ . in Eq (4).

In Figure 1 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.

#### 4. Estimating $V_m$ from planning solutions

RetRO\* requires the value function oracle  $V_m$  to compute  $V_t(\cdot|T)$  for 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.

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 Appendix E.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  $(R_j, \mathcal{S}_j, c(R_j)) \in B(m_i)$ :

$$\mathcal{L}_{con}(rt_i, R_j) = \max \left\{ 0, v_i + \epsilon - c(R_j) - \sum_{m' \in \mathcal{S}_j} V_{m'} \right\} \quad (6)$$

where  $\epsilon$  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:

$$\min_{V(\cdot)} \mathbb{E}_{rt_i \sim \mathcal{R}_{train}} \left[ \mathcal{L}_{reg}(rt_i) + \lambda \mathbb{E}_{R_j \sim B(m_i) \setminus \{R_i\}} [\mathcal{L}_{con}(rt_i, R_j)] \right] \quad (7)$$

where  $\lambda$  (default:  $\lambda = 1$ ) balances these two losses.

## 5. Experiments

### 5.1. Results on small molecules

We compare RetRO\* against DFPN-E [11], MCTS [18] and greedy Depth First Search (DFS) on product molecules in test route dataset described in Appendix E.1.2. We use route quality and planning efficiency to evaluate the algorithm.

**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.

**Influence of time limit:** To show the influence of time limit on performance, we plot the success rate against the

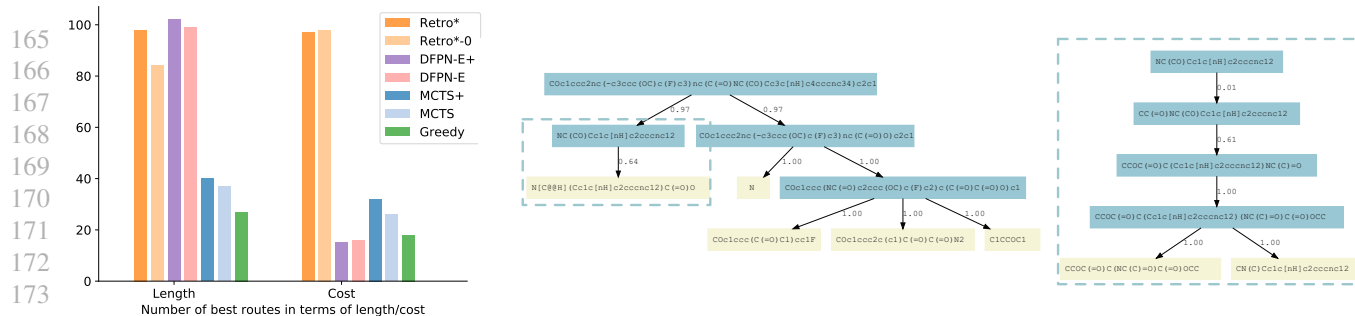


Figure 2. **Left:** Counts of the best solutions among all algorithms in terms of length/cost; **Mid:** Sample solution route from Retro\*. Numbers on the edges are the likelihoods of the reactions. Yellow nodes are building blocks; **Right:** The corresponding dotted box part in the expert route, much longer and less probable than the solution.

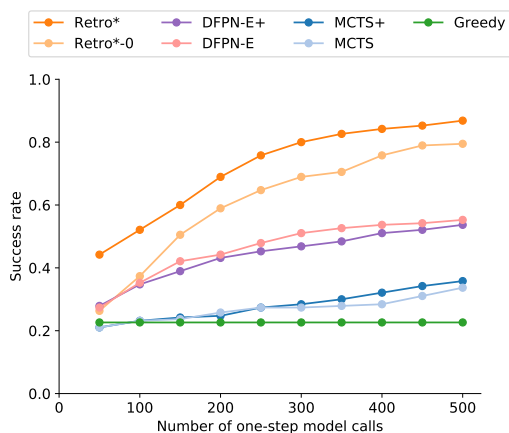


Figure 3. Influence of time limit on performance.

number of one-step model calls in Figure 3. 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 2-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 optimizing for, it still achieves about the same performance as the best method.

**Ablation study:** We also conduct an ablation study to understand the importance of the learning component in Retro\* by evaluating its non-learning version Retro\*-0, learning-strengthened baselines MCTS+ and DFPN-E+. Please refer to Appendix E.2 for more discussions.

As a demonstration for Retro\*'s ability to find high-quality routes, we illustrate a sample solution in Figure 2-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 2-right) is much longer and less probable than the corresponding part of the solution route, as shown in the dotted box region in Figure 2-mid. Please refer to Appendix F for more sample solution routes and search tree visualizations.

## 5.2. Results on polymers

Retrosynthesis for polymers shares roughly the same setting as small molecules, except that additional structural constraints are imposed on the first search step, the polymerization reaction. Details on the additional constraints, the problem formulation of polymer retrosynthesis, the adapted polymer induction technique for dealing with structural constraints, and PolyRetro are presented in Appendix I.

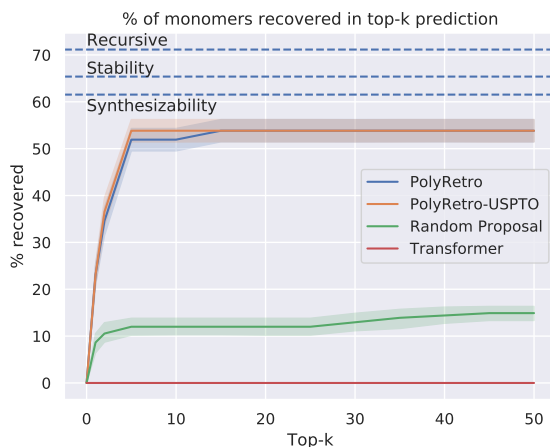


Figure 4. Percentages of monomers recovered in top-k prediction.

We evaluate the performance of PolyRetro in a real-world polymer dataset. The monomer recovery result in Figure 4 demonstrate our method's dominating performance over all baselines. More details are presented in Appendix J.



## References

- [1] Rdkit: Open-source cheminformatics. URL <http://www.rdkit.org>.
- [2] Allis, L. V., van der Meulen, M., and Van Den Herik, H. J. Proof-number search. *Artificial Intelligence*, 66(1):91–124, 1994.
- [3] 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.
- [4] Chen, B., Li, C., Dai, H., and Song, L. Retro\*: Learning retrosynthetic planning with neural guided a\* search. In *The 37th International Conference on Machine Learning (ICML 2020)*, 2020.
- [5] 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.
- [6] 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.
- [7] Erol, K. *Hierarchical task network planning: formalization, analysis, and implementation*. PhD thesis, 1996.
- [8] 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.
- [9] 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.
- [10] Karpov, P., Godin, G., and Tetko, I. A transformer model for retrosynthesis. 2019.
- [11] 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.
- [12] Kocsis, L. and Szepesvári, C. Bandit based Monte-Carlo planning. In *European conference on machine learning*, pp. 282–293. Springer, 2006.
- [13] 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.
- [14] Rogers, D. and Hahn, M. Extended-connectivity fingerprints. *Journal of chemical information and modeling*, 50(5):742–754, 2010.
- [15] Schreck, J. S., Coley, C. W., and Bishop, K. J. Learning retrosynthetic planning through simulated experience. *ACS Central Science*.
- [16] 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.
- [17] 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.
- [18] Segler, M. H., Preuss, M., and Waller, M. P. Planning chemical syntheses with deep neural networks and symbolic ai. *Nature*, 555(7698):604, 2018.
- [19] 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.
- [20] 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.
- [21] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.
- [22] Yang, K. and Deng, J. Learning to prove theorems via interacting with proof assistants. *arXiv preprint arXiv:1905.09381*, 2019.

## A. More Backgrounds

### A.1. One-step retrosynthesis

Existing one-step retrosynthesis 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 [5], neuralsym [17] and GLN [6]. 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 [13] or Transformer [10] from neural machine translation literature.

However, MCTS-based methods has several limitations in this setting:

- Each tree node corresponds to a set of molecules instead of single molecule. This additional combinatorial aspect make the representation of tree node, and the estimation of its value even harder. Furthermore, reactions do not explicitly 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 retrosynthesis 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.

### A.2. Monte Carlo Tree Search

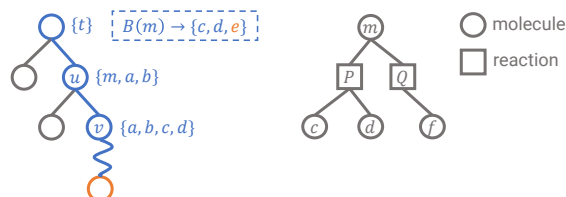


Figure 5. **Left:** MCTS [18] for retrosynthesis planning. Each node represents a set of molecules. Orange nodes/molecules are available building blocks; **Right:** AND-OR stump illustration of  $B(m) = P, Q$ . Reaction  $P$  requires molecule  $c$  and  $d$ . Reaction  $Q$  requires molecule  $f$ . 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 [19, 20]. Its variant, UCT [12], is especially powerful for balancing exploration and exploitation in online learning setting, and has been employed in Segler et al. [18] for retrosynthesis planning. Specifically, as illustrated in Figure 5, 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 rollout 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.

### A.3. Proof Number Search and Variants

The proof-number search (PNS) [2] 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  $u$ ,

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

and for proved nodes:  $pn(u) = 0, dn(u) = +\infty$  (8)

and for OR node  $u$ , we have

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

and for disproved node:  $pn(u) = +\infty, dn(u) = 0$  (9)

**Represent retrosynthesis planning using AND-OR tree:** As illustrated in Figure 5, 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 [11] builds on top of the depth-first variant of PNS with an additive cost in addition to classical update rule in Eq (9). Specifically, for an unsolved OR node,

$$pn(u) = \min_{v \in ch(u)} (h(u, v) + pn(v))$$
 (10)

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 dealing with new one-step model or reaction data.

Our proposed `Retros*` 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:

- `Retros*` 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 `Retros*` advantageous in finding best retrosynthetic routes.
- `Retros*` 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. [18], or the human designed heuristics in Kishimoto et al. [11]. This not only enables more accurate prediction during expansion, but also generalizes the knowledge learned from existing planning paths.

## B. More details of Retro\*

### B.1. Retro\* algorithm

---

**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    $m_{next} \leftarrow \operatorname{argmin}_{m \in \mathcal{F}(T)} V_t(m)$ ;
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;
```

---

### B.2. 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 (4), 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 (4). 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 (5) 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 C.

### B.3. 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 (4) 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.

### B.4. Representation of $V_m$

To parameterize  $V_m$  for any molecule  $m$ , we first compute its Morgan fingerprint [14] 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  $V_m$ .



## C. Implementation details in the update phase

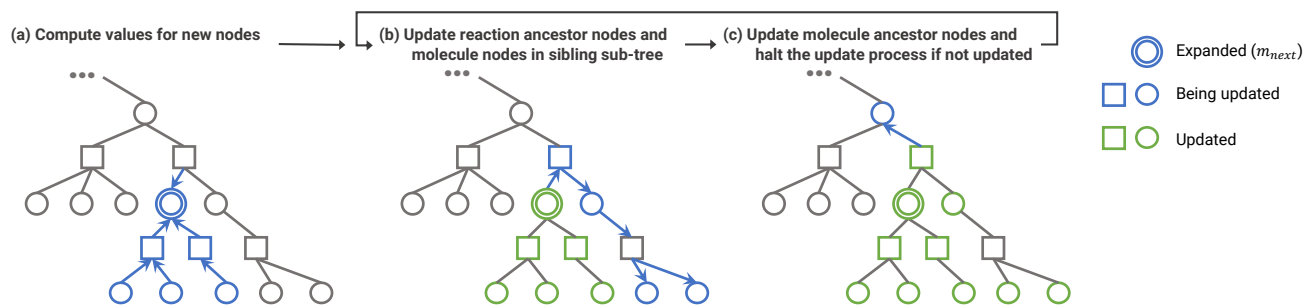


Figure 6. Illustration for the update process. Three phases correspond to line 1-8, line 11-16, and line 17-21 in Algorithm 2.

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 (5), 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:**  $\text{Update}(m_{next}, \{R_i, \mathcal{S}_i, c(R_i)\}_{i=1}^k)$ <sup>1</sup>

---

```

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  $new\_rn \leftarrow \min_{i \in \{1, 2, \dots, k\}} rn(R_i)$ ;
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       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 6, 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

<sup>1</sup>For clarity, we omit the condition on  $T$  in the notations.

reactions according to Eq (4) and (5) 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 (4) and (5), avoiding a direct computation which may require  $O(k)$  or  $O(\text{depth}(T))$  summations.

## D. 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  $\operatorname{argmin}_{m \in \mathcal{F}(T)} V_t(m)$ ”.

**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.

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

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 (3),  $V_t(m|T)$  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  $V_{m_s}$ . 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 `Retros*` is a lowerbound of true  $h_t(m|T)$ .

We prove it with existing results in A\* literature.

**Proof** Combine Lemma 2 and Theorem 1 in the original A\* paper [9]. ■

## E. Retro\* experiment details

### E.1. Creating benchmark dataset

#### E.1.1. USPTO REACTION DATASET

We use the publicly available reaction dataset extracted from United States Patent Office (USPTO) to train one-step model 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 RDChiral<sup>2</sup> 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 [17] 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.

#### E.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*<sup>3</sup>. *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.

### E.2. Detailed results

We compare Retro\* against DFPN-E [11], MCTS [18] and greedy Depth First Search (DFS) on product molecules in test route dataset described in Appendix E.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 efficiency, 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

<sup>2</sup><https://github.com/connorcoley/rdchiral>

<sup>3</sup><http://downloads.emolecules.com/free/2019-11-01/>

## Learning Retrosynthetic Planning with Chemical Reasoning

| 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.

conduct an ablation study to understand the importance of the learning component in  $\text{Retro}^*$  by evaluating its non-learning version  $\text{Retro}^*-0$ .  $\text{Retro}^*-0$  is obtained from  $\text{Retro}^*$  by setting  $V_m$  to 0, which is a lowerbound of any valid values. Comparing to baseline methods,  $\text{Retro}^*-0$  is also showing promising results. However, it is outperformed by  $\text{Retro}^*$  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  $\text{Retro}^*$ , 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 don't have a good initialization of the disproof number.

**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 3. We can see that  $\text{Retro}^*$  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 2-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,  $\text{Retro}^*$  produces 4 $\times$  more best routes than the second best method. Even for the length metric, which is not the objective  $\text{Retro}^*$  is optimizing for, it still achieves about the same performance as the best method.

As a demonstration for  $\text{Retro}^*$ 's ability to find high-quality routes, we illustrate a sample solution in Figure 2-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 2-right) is much longer and less probable than the corresponding part of the solution route, as shown in the dotted box region in Figure 2-mid. Please refer to Appendix F for more sample solution routes and search tree visualizations.



## F. 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 7, 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.

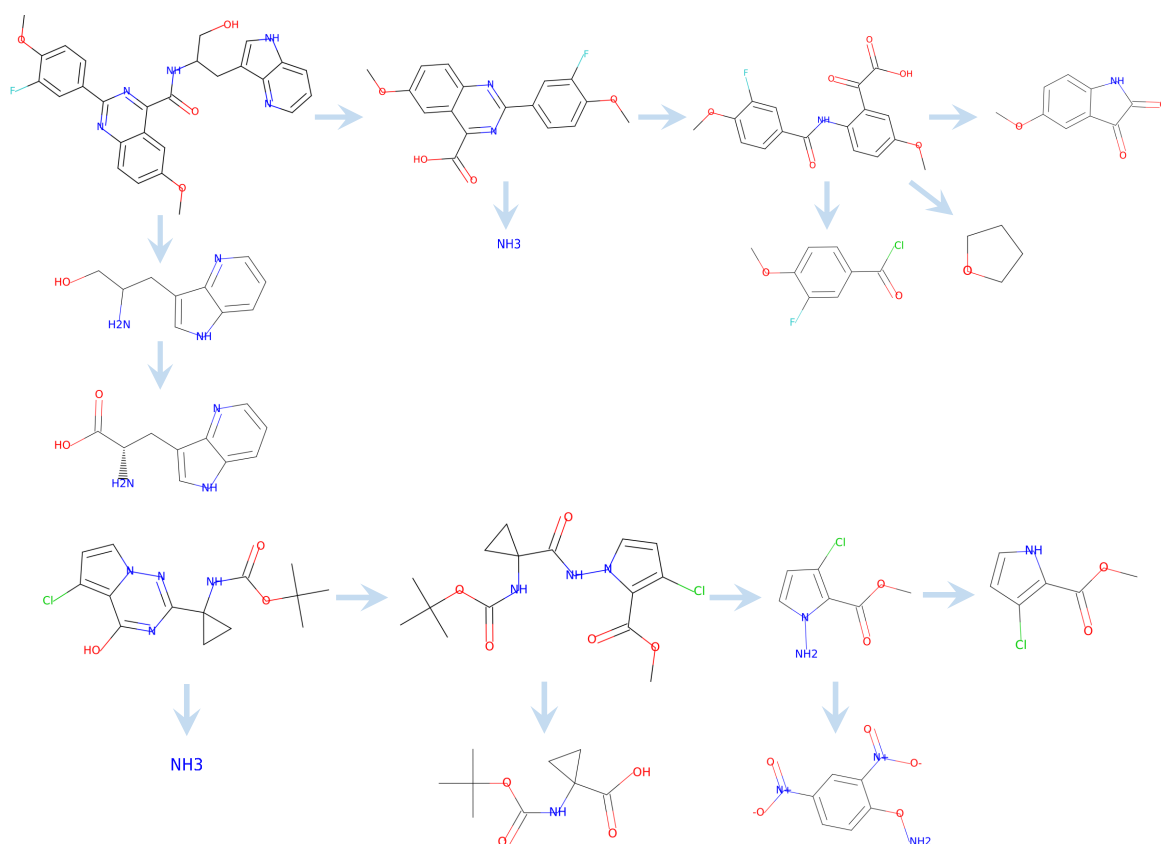


Figure 7. Top/bottom: solution route produced by *Retro\** for molecule *A/B*. Edges point from the same product molecule to the reactant molecules represent an one-step chemical reaction.

The search trees for molecule *A* and *B* are illustrated in Figure 8 and Figure 9. We use rectangular boxes to represent molecules. Yellow/grey/blue boxes indicate available/unexpanded/solved molecules. Rectangular 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.

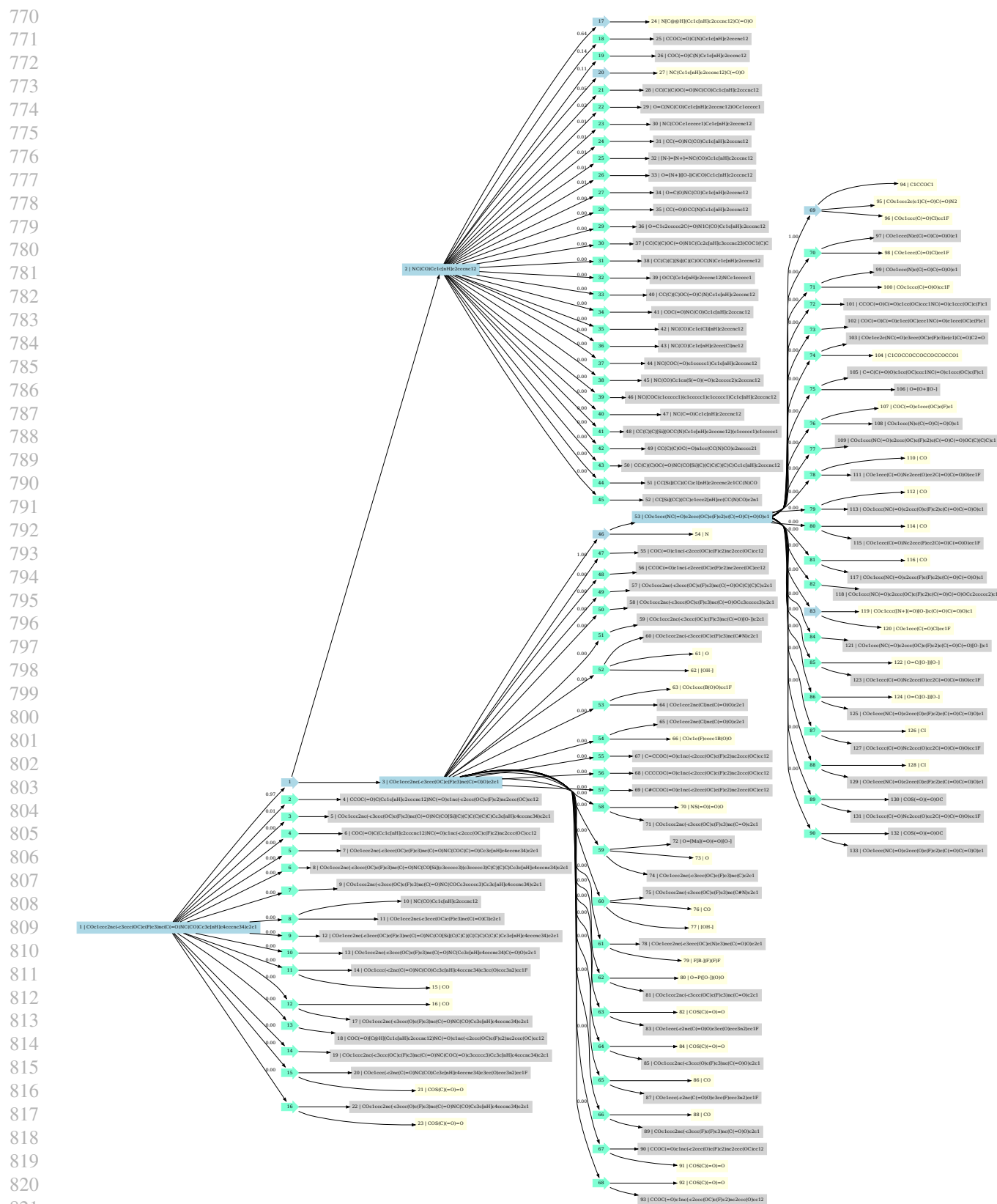


Figure 8. Search tree produced by RETRO\* for molecule A. Rectangular boxes/arrows represent molecules/reactions. Yellow/grey/blue indicate available/unexpanded/solved molecules. Numbers on the edges are the probabilities produced by the one-step model.

825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879

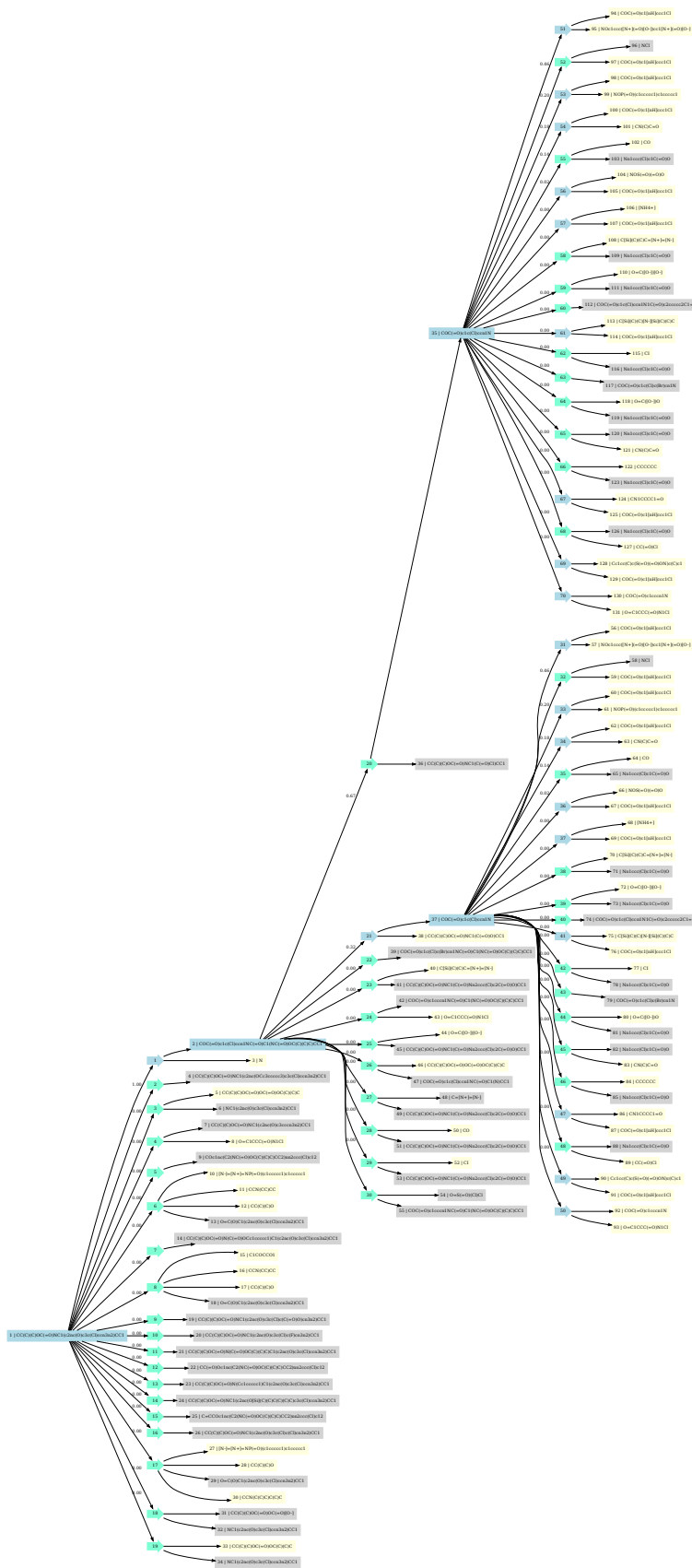


Figure 9. Search tree produced by Retrosyn\* for molecule B. Rectangular boxes/arrows represent molecules/reactions. Yellow/grey/blue indicate available/unexpanded/solved molecules. Numbers on the edges are the probabilities produced by the one-step model.

## G. Retro\* for hierarchical task planning

As a general planning algorithm, Retro\* can be applied to other machine learning problems as well, including theorem proving [22] and hierarchical task planning [7] (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.

| 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.

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

## H. More Related Works

Reinforcement learning algorithms (without planning) have also been considered for the retrosynthesis problem. Schreck et al. [15] 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 [19, 20], Sokoban [8] and path planning [3]. 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.



## I. Polymer retrosynthesis with PolyRet ro

### I.1. Background

In this section, we focus on providing the background knowledge about molecule retrosynthesis as well as defining notations. This serves as the building block in our polymer retrosynthesis modeling.

Given a molecule  $m \in \mathcal{M}$  where  $\mathcal{M}$  indicates the space of molecules, the molecule retrosynthesis problem focuses on finding a set of reactants  $\mathcal{S} \subset \mathcal{M}$  that can be used to synthesize  $m$ . Before introducing the approaches for retrosynthesis, we first cover the background on reaction templates.

#### I.1.1. REACTION TEMPLATE

A reaction template  $T := o^T \rightarrow r_1^T + r_2^T + \dots + r_{|T|}^T$  is a graph rewriting rule<sup>4</sup> that rewrites subgraph pattern  $o^T$  that is matched with target molecule  $m$ , into  $r_i^T$  that appears in  $i$ -th reactant  $s_i \in \mathcal{S}$ . The set of templates  $\mathcal{T}$  can be extracted from existing chemical reactions in the literature. Although applying templates involves with expensive subgraph matching between  $o^T$  and  $m$ , where itself is an NP-hard problem, such approach provides a tractable way of finding candidate set  $\mathcal{S}$  with chemical rules.

#### I.1.2. LEARNING-BASED MOLECULE RETROSYNTHESIS

The molecule retrosynthesis problem has raised increasing interest in the machine learning (ML) community, due to its importance in chemistry and the difficulty in structured prediction setting. We here mainly focus on the ML approaches for such problem, as some of these provide probabilistic interpretations that will be needed in our optimization framework. Depends on the number of reaction steps needed to synthesize  $m$ , such problem can be categorized into one-step and multi-step retrosynthesis.

**One-step molecule retrosynthesis:** One-step setting requires that  $R := \mathcal{S} \rightarrow m$  can be realized in one chemical reaction. It focuses on modeling  $p(\mathcal{S}|m)$  with or without reaction templates. As the template based one guarantees the satisfaction of human defined rules, we use the model proposed in NeuralSym [17] for one-step prediction. Specifically in this model:

$$p(\mathcal{S}|m) \propto \sum_{T \in \mathcal{T}} p(T|m) \mathbb{I}[\text{SubgMatch}(o^T, m)] \quad (11)$$

where  $\text{SubgMatch}(\cdot)$  operator checks the subgraph matching between  $o^T$  and  $m$ .

**Multi-step molecule retrosynthesis** The multi-step extension allows using multiple reactions  $\mathcal{R}_m = \{R_i^m\}_{i=1}^{|\mathcal{R}_m|}$  to synthesize  $m$ , with the restriction that the reactants set  $\mathcal{S} \subset \mathcal{I} \subset \mathcal{M}$  where  $\mathcal{I}$  is the set of starting molecules. This is essentially a planning problem that search through the reaction space using one-step models as expansion proposals. In our paper, we use the Retro\*[4] which is the state-of-the-art approach that provably optimizes the synthesizability of  $\mathcal{R}_m$ .

### I.2. Modeling Polymer Retrosynthesis

A polymer is a large molecule composed of many repeat units. Directly using molecule retrosynthesis techniques for polymers is not feasible, as (i) the (potentially infinite) large molecule is not feasible for existing molecule retrosynthesis approaches; (ii) the polymer retrosynthesis has nontrivial recursive and stability constraints, which cannot be easily addressed in existing approaches.

In the following content, we first state these constraints in Section I.2.1, then we formally present our modeling strategy for such problem in Section I.2.2.

#### I.2.1. CONSTRAINTS FOR POLYMERS AND POLYMERIZATIONS

In chain structured polymer, each repeat unit in the polymer has two open bonds which link with neighboring units in a chain-like structure. At the end of the chain are the end-groups. They are functional groups closely related to the polymerization process, where the polymer is synthesized from the monomers.

In this work we focus on **condensation polymerization**, in which large molecules join together and lose small molecules such as  $\text{H}_2\text{O}/\text{HCl}$  (Figure 10). This type of polymerization needs to satisfy the recursive constraint (unit polymerization)

<sup>4</sup><https://www.daylight.com/dayhtml/doc/theory/theory.smarts.html>

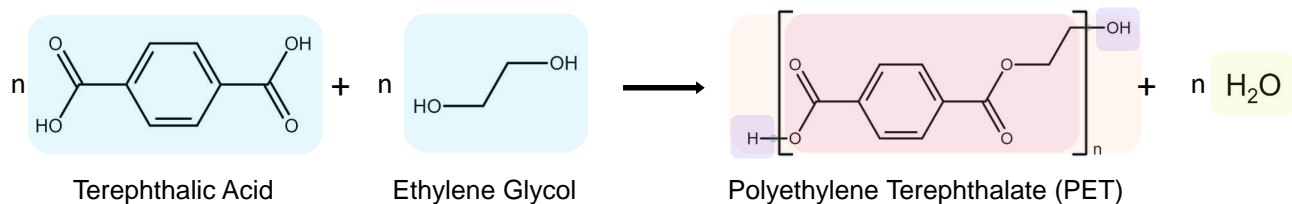
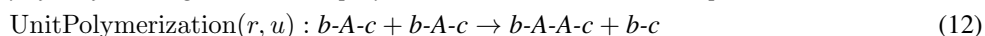


Figure 10. Condensation polymerization for synthesizing PET. We use shaded color regions to highlight the monomers (light blue), polymer (light pink), repeat unit (dark pink), end-groups (purple), and by-product (light yellow). Unit polymer is a polymer with unit length ( $n = 1$ ).

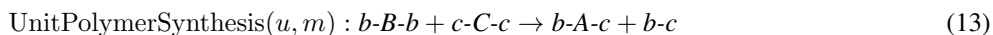
placed on the unit polymer.

**Definition 3 (Recursive constraint)** Given the repeat unit  $r$  with structure  $-A-$ , where we use  $'-'$  to represent an open bond, which connects to the neighboring repeat unit in a long chain, the unit polymer  $u$  should be  $b-A-c$ , where  $b-$  and  $-c$  are end-groups, and the probability of the following induced unit polymerization reaction should be positive:



In this reaction, two unit polymers join together and lose  $b-c$  as the byproduct. In practice, we hope that the end-groups  $b-$  and  $-c$  tend to react with each other with high probability, which makes the polymerization reaction continue to happen. Therefore the unit polymer itself is not stable. However the monomers, which are the precursors of the unit polymer, should satisfy the stability constraint.

**Definition 4 (Stability constraint)** Given the unit polymer  $u$  with structure  $b-A-c$ , the monomers  $m$  should be  $\{b-B-b, c-C-c\}$ , where  $b-B-b$  and  $c-C-c$  are symmetric, and  $u$  can be synthesized from  $m$  with the following reaction:



Both the recursive and the stability constraint come from chemical insights which guide polymer synthesis.

## 1.2.2. POLYMER RETROSYNTHESIS MODELING

Section 1.2.1 has defined the structural constraints for polymer retrosynthesis, which are symbolic with rule definitions. In real application, polymerization also requires the efficiency to make the entire pipeline cost efficient. With both the rule constraints and efficiency requirement, we formulate the polymer retrosynthesis as a constrained optimization problem. We first define molecule synthesizability below:

**Definition 5 (Molecule synthesizability)** A molecule  $m$  is  $h$ -synthesizable from a set of starting molecules  $\mathcal{I}$  if and only if there exists a series of reactions  $\mathcal{R}$  whose initial reactants are in  $\mathcal{I}$ , and the joint probability of reactions in  $\mathcal{R}$  satisfy a given lower bound  $h$ , i.e.  $p(\mathcal{R}) > h$ .

Practically we rely on the molecule retrosynthesis algorithm presented in 1.1.2 to obtain both  $\mathcal{R}$  and  $p(\mathcal{R})$ . Using the key concepts defined above, we state our optimization formulation of polymer retrosynthesis problem below.

**Definition 6 (Polymer Retrosynthetic Optimization)** Given a target polymer represented as a repeat unit  $r$ , we want to identify a unit polymer  $u$ , and monomers  $m_1, m_2$  which maximize the polymerization probability with the recursive, stability, and molecule synthesizability constraints.

$$\begin{aligned} \max_{u, m_1, m_2} \quad & p(u, m_1, m_2 | r) = p(u | r) \cdot p(m_1, m_2 | u) \\ \text{s. t.} \quad & p(\mathcal{R}_{m_1, m_2}) > h. \end{aligned} \quad (14)$$

with a positive threshold  $h$ .

The model  $p(u, m_1, m_2 | r)$  is decomposed into  $p(u | r)$  that incorporates the recursive constraint, and  $p(m_1, m_2 | u)$  that incorporates the stability constraint. The formulation can be interpreted using a mathematical induction analogy. Synthesizing the target polymer can be understood as proving the feasibility of the polymer. To “prove” the target polymer, we need to first work on the base case, i.e., find a monomer which is synthesizable, and then prove the induction step, i.e., maximize the polymerization probability.

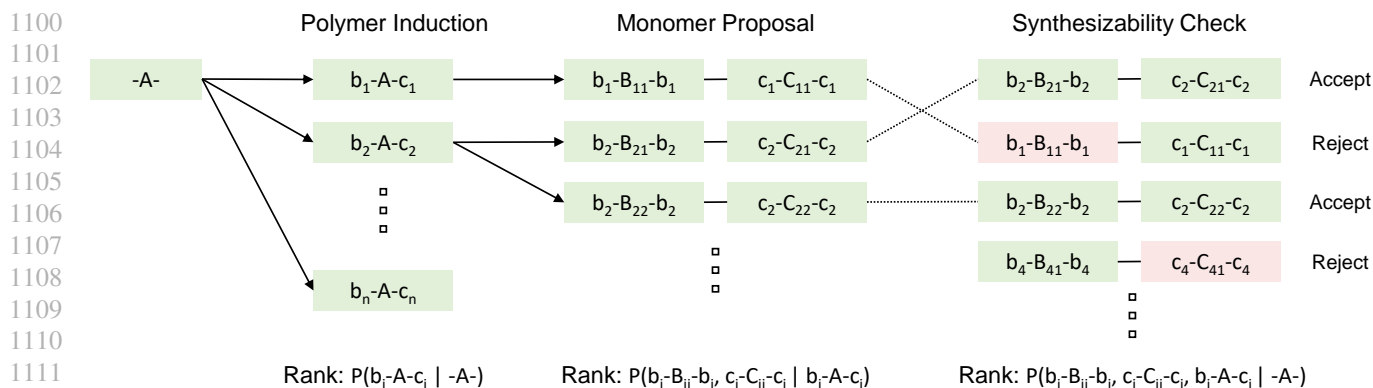


Figure 11. PolyRetro beam search framework. Given the repeat unit  $r$ , we first generate unit polymer candidates  $u$  by polymer induction. We then rank the candidates using  $p(u|r)$  and sample monomers  $(m_1, m_2)$  from the top- $n$  unit polymers. Next we re-rank the unit polymer and monomer pair  $(u, m_1, m_2)$  using the joint  $p(u, m_1, m_2|r) = p(u|r) \cdot p(m_1, m_2|u)$  and perform the synthesizability check on  $(m_1, m_2)$  using Retro\*, a multi-step retrosynthesis planner. Finally the top- $k$  results which pass the check are returned.

Although  $p(m_1, m_2|u)$  can be characterized using one-step molecule retrosynthesis model, directly optimizing the above problem is nontrivial, as (i) the recursive constraint model  $p(u|r)$  which predicts a unit polymer from a repeat unit is not available and there is not enough data in the literature to estimate such model; and (ii) the constraint of  $p(\mathcal{R}_{m_1, m_2}) > h$  goes through a planning algorithm that is not possible to characterize the gradient, convexity, etc.. In the next section, we will present PolyRetro to effectively solve the constrained optimization by overcoming these difficulties.

### I.3. PolyRetro algorithm

To tackle the challenging constrained optimization problem defined in Section I.2.2, we propose PolyRetro (Figure 11), a learning-based search framework with minimal polymerization data involved. We first introduce the overall procedures of the algorithm, and then cover the details in subsections.

Our main idea is based on the rejection sampling framework for solving Eq (14), which treat the molecule synthesizability constraint as a black-box rejection criteria. Although asymptotically we can sample target solution  $(u, m_1, m_2)$  from Eq (14), a good proposal algorithm is needed to keep the rejection rate low and mix fast. Since the joint probability of  $(u, m_1, m_2)$  decomposes into two terms, we approximate the max operator with a beam search in two steps:

$$\text{top-}k_{u, m_1, m_2} p(u, m_1, m_2|r) \simeq \text{top-}k_{u, m_1, m_2} p(m_1, m_2|u) \mathbb{I}[u \in \text{top-}n \{p(u|r)\}] \quad (15)$$

We present the modeling and inference of these two steps in the next two sections. In Section I.3.1 we show how to generate top- $n$  candidate unit polymers that satisfy with recursive constraint using domain adaptation with a novel *polymer induction* technique. Then in Section I.3.2, we show the generative procedure of monomers with stability constraints. We conclude the approach with the molecule synthesizability check in Section I.3.3.

#### I.3.1. UNIT POLYMER PROPOSAL WITH DOMAIN ADAPTED POLYMER INDUCTION

In the first step of the beam search, we want to quickly generate all the feasible candidate unit polymers from the repeat unit  $r$ . Proposing candidates that satisfy recursive constraint is hard. This involves predicting the end-groups from the repeat units. Also note that there will not be enough data to directly learn the model  $p(u|r)$ .

We observe that, it is relatively easy to obtain the one-step retrosynthesis model (one-step model for short)  $p(S|m)$  in Eq (11) for a molecule  $m$  and there are enough small molecule chemical reactions to train such one-step model. Based on this, we propose to perform *domain adaptation* using *polymer induction* technique (Figure 12), as explained below:

**Polymer induction:** the basic idea is to leverage one-step molecule retrosynthesis model to help predict the end groups of a unit polymer. As the model  $p(S|m)$  only accepts molecules rather than repeat units, we circumvent this issue with the following procedure:

1. Link two repeat units head-to-tail to form a double repeat unit, and add hydrogen as end-groups, i.e. H-A-A-H;
2. Loop through all the reaction templates:
  - (a) Apply reaction template to the double repeat unit;

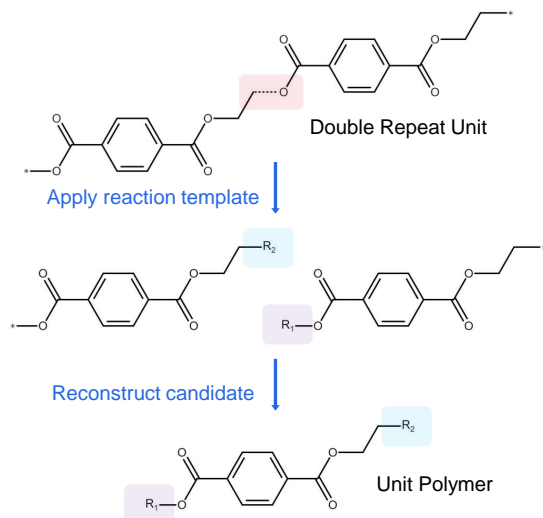


Figure 12. Polymer induction for generating unit polymer candidates. For each of the reaction template, we apply it to the double repeat unit, if it can break the bond connecting two repeat units into two end-groups. These two end-groups are then being put back to the repeat unit to reconstruct unit polymer candidates.

(b) If the result is in the form of  $\text{H-A-A-H} \rightarrow \text{H-A-c} + \text{b-A-H}$ , add  $\text{b-A-c}$  to the candidate list.

The above procedure is based on the induction principle: if the base case solves (*i.e.*, we find the unit polymer with form  $\text{b-A-c}$ ), then we can synthesize the double units  $\text{b-A} \sim \text{A-c}$  where the bond between two As (denoted as  $\sim$  symbol) belongs to the reaction center. Such induction thus provides the guidance for the template based retrosynthesis: a template  $T = o^T \rightarrow r_1^T + r_2^T$  should have the subgraph  $o^T$  matched at the location that covers the bond denoted as  $\sim$ . Using template based graph rewriting with  $r_1^T$  and  $r_2^T$ , we can obtain the end groups  $b$  and  $c$ .

**Modeling  $p(u|r)$  with domain adaptation:** the polymer induction step gives us a list of unit polymer and corresponding template candidates  $\{(T_i, b_i\text{-A-}c_i)\}$  that satisfy the recursive constraint, which is actually the support of  $p(u|r)$  (*i.e.*, samples with non-zero probability). Given a unit polymer and template pair ( $u = \text{b-A-c}$ ,  $T$ ), we formulate the corresponding joint probability as the optimal solution of following optimization:

$$\min_{p(u=\text{b-A-c}, T|r)} \underbrace{\lambda D_{KL}(p(\mathcal{S} = \{u, u\} | m = \text{b-A-A-c}) || p(u, T|r))}_{\text{matching one-step model}} + (1 - \lambda) \underbrace{D_{KL}(\hat{p}(T) || p(u, T|r))}_{\text{matching target domain prior}} \quad (16)$$

where  $\hat{p}(T)$  is an empirical estimation that of template prior from the limited polymer dataset, and  $\lambda \in (0, 1)$  is a factor that balances the two Kullback–Leibler divergences. The optimal solution is  $p(u = \text{b-A-c}, T|r) = \lambda p(\{u, u\} | \text{b-A-A-c}) + (1 - \lambda)\hat{p}(T)$ . This objective performs the domain adaptation from molecule synthesis into polymer synthesis with the following benefits:

- With the polymer induction technique, we adapt one-step model for molecules to polymer domain.
- By interpolating between one-step model and the prior learned on target domain, we can achieve a balance between enormous foreign domain knowledge and limited target-domain priors.

There could be multiple design choices for  $\hat{p}(T)$ . Due to the limited data in target (polymer) domain, we use kernel density estimation with atom- and bond-counting features for simplicity.

### I.3.2. MONOMER PROPOSAL UNDER STABILITY CONSTRAINT

Using domain adapted polymer induction in the above section, we can obtain top- $n$  unit polymers  $\{b_i\text{-A-}c_i\}_{i=1}^n$  and their scores. The second step of the beam search seeks for monomers given the unit polymers. As both of them are proper molecules, the one-step model can be directly used here:

$$p(m_1, m_2|u) = p(\mathcal{S} = \{m_1, m_2\} | m = u) \mathbb{I}[\text{Both } m_1 \text{ and } m_2 \text{ are symmetric}] \quad (17)$$

Adding together with the scores of each unit polymer  $u$ , we can approximately get a list of feasible unit polymer and monomers with the highest joint probability.

## 1210 I.3.3. SYNTHESIZABILITY AS FILTERING CRITERIA

1211 Once we have the top- $n$  list of monomers  $\{m_1^i, m_2^i\}_{i=1}^n$  obtained in previous two beam-search steps, we can perform  
1212 rejection step using any off the shelf multi-step retrosynthesis planner to check the synthesizability  $p(\mathcal{R}_{m_1, m_2})$  for each  
1213  $(m_1, m_2) \in \{m_1^i, m_2^i\}_{i=1}^n$ , and return the top- $k$  of them which satisfies the recursive, stability and synthesizability constraints.  
1214 This finishes the overall `PolyRetro` algorithm.  
1215  
1216  
1217  
1218  
1219  
1220  
1221  
1222  
1223  
1224  
1225  
1226  
1227  
1228  
1229  
1230  
1231  
1232  
1233  
1234  
1235  
1236  
1237  
1238  
1239  
1240  
1241  
1242  
1243  
1244  
1245  
1246  
1247  
1248  
1249  
1250  
1251  
1252  
1253  
1254  
1255  
1256  
1257  
1258  
1259  
1260  
1261  
1262  
1263  
1264



## J. PolyRetro experiment

**Experiment Settings.** To evaluate the performance of PolyRetro, we collect a dataset of 52 condensation polymers<sup>5</sup>, and their corresponding synthesis recipes. For each polymer, the task is to predict the ground truth monomers and unit polymer given the repeat unit. We split the data into 5 folds and repeat the same experimental setup 5 times to compute the mean and standard deviation of the performance metric. Because we are targeting a few-shot learning setting and considering we do not have enough data for testing, in each experiment, we only use one fold of data for training and hold out the other four folds for evaluation. We use top- $k$  recovery rate, i.e. whether the target is in the top- $k$  prediction, on both unit polymers and monomers to measure the performance of the algorithms.

**PolyRetro Implementation.** The one-step retrosynthesis model used in PolyRetro is trained on the publicly available reaction dataset extracted from United States Patent Office (USPTO). We use a multilayer perceptron model with one hidden layer of size 512, which takes a 2048 dimensional molecular fingerprint [14] as input, and predicts a probability distribution over 230k reactions templates extracted from the same dataset. Reactants can be obtained by applying the reaction templates to the product using RDKit [1]. In the monomer retrosynthesis step, Retro\* [4] employs the same model for synthesis route search given a list of commercially available building blocks from eMolecules<sup>6</sup>, which consists of 231M commercially available molecules.

We implement the template prior in PolyRetro using a nonparametric Gaussian kernel density estimator with bandwidth 1. Its performance is not sensitive to the bandwidth parameter. The features are the number of atoms and edges on both side of the template. We set  $\lambda$  to 0.999 in Eq (16).

**Baselines/Ablation Studies.** We compare PolyRetro with a Transformer [21] model which directly predicts the target sequence (unit polymers/monomers) from the repeat unit. We also conduct an ablation study by replacing the unit polymer proposal step in PolyRetro with (a) sampling randomly from the support of  $p(u|r)$  (Random Proposal), and (b) using only the one-step retrosynthesis model trained on USPTO as  $p(u|r)$  (PolyRetro-USPTO). Since learning a sequence-to-sequence model from only a few data points is nearly impossible, we give all Transformer-based models a large advantage by allowing the use of the test data for validation for early stopping in training.

**Results.** The top- $k$  recovery rate result on unit polymers and monomers is summarized in Figure 13. PolyRetro achieves over 50% top-1, 71% top-30 recovery rate in unit polymer prediction, and over 50% top-5 recovery rate in monomer prediction. The performance is significantly better than the sequence-to-sequence baseline, which is not able to learn anything, achieving 0% top-50 recovery rate in both results, not to mention that the baseline model can see the test set during training. This is within our expectation due to the lack of training data. With more data we should see an increase in the number. However, end-to-end approaches still suffer from the inherent limitation of failing to address the structural constraints.

**Domain Adaptation.** Knowledge transfer from small molecule reactions to unit polymerization is clearly supported by the result, as PolyRetro and PolyRetro-USPTO outperform the random proposal baseline by a large margin. With such strong performance of PolyRetro recovering the ground truth monomers, we have reason to believe the other monomers generated by our algorithm could also be of realistic significance. Domain adaptation by incorporating template prior is also helpful for improving the performance. With the template prior learned on one fold of data, PolyRetro is able to have a 3 – 5% gain in performance compared with PolyRetro-USPTO when predicting unit polymers. The main reason that PolyRetro is not improving the monomer prediction further is because of the limitation on unseen templates for one-step retrosynthesis, which we discuss below.

**Limitations.** We display the performance upper bound of PolyRetro in the monomer prediction result in Figure 13. From top to bottom, the three horizontal lines correspond to the upper bounds after addressing the recursive constraint, the recursive and stability constraints, and all three constraints. These upper bounds are resulted from the following limitations of the algorithm.

- **Constraint Formulation.** Our constraint formulations in Definition 3 and 4 describe the most common pattern for polymerization but unable to cover all special cases. The algorithm can be improved by incorporating more chemical knowledge.
- **Unseen Reaction Templates.** Since the templates for one-step retrosynthesis model are extracted from the small molecule

<sup>5</sup><http://www.polymerdatabase.com/polymer%20chemistry/pc%20index.html>

<sup>6</sup><http://downloads.emolecules.com/free/2019-11-01/>

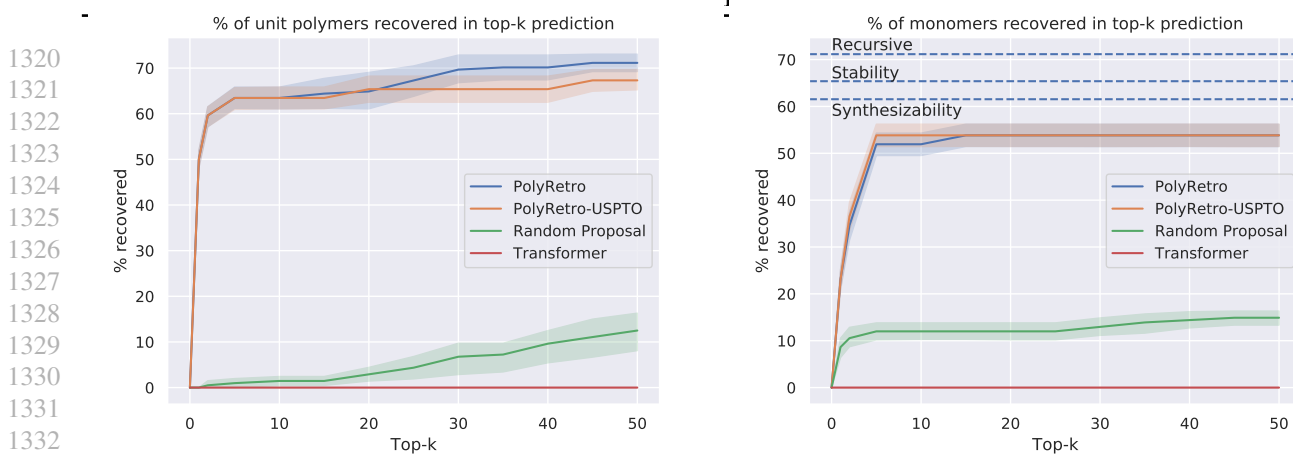


Figure 13. Percentages of targets recovered by algorithms in top- $k$  prediction. Left: unit polymers prediction using  $p(u|r)$ ; Right: monomer prediction using  $p(u|r) \cdot p(m_1, m_2|u)$ . We also show PolyRetro's performance upper bounds after addressing the labeled constraints on the right.

reactions, there could be new reaction templates which only exist in polymerization. The limitation can be alleviated by extracting templates and training the one-step retrosynthesis model directly using polymerization data.

- **Monomer Retrosynthesis.** For rare cases, Retro\* cannot find a synthesis path for the ground truth monomer in limited time. This can be resolved by allowing more time for Retro\* search and using a larger building block molecule set.