Received: 26 September 2023 | Revised: 19 February 2024 | Accepted: 26 March 2024
DOI: 10.1002/minf.202300259

RESEARCH ARTICLE

molecular informatics

Comparing search algorithms on the retrosynthesis problem

Milo Roucairol | **Tristan Cazenave**

LAMSADE, Université Paris Dauphine - PSL, Paris, France

Correspondence
Milo Roucairol, LAMSADE, Université Paris Dauphine - PSL, Paris, France.
Email: milo.roucairol@dauphine.eu

Funding information
French government under the management of Agence Nationale de la Recherche, Grant/Award Number: ANR19-P3IA-0001

Abstract
In this article we try different algorithms, namely Nested Monte Carlo Search and Greedy Best First Search, on AstraZeneca's open source retrosynthetic tool : AiZynthFinder. We compare these algorithms to AiZynthFinder's base Monte Carlo Tree Search on a benchmark selected from the PubChem database and by Bayer's chemists. We show that both Nested Monte Carlo Search and Greedy Best First Search outperform AstraZeneca's Monte Carlo Tree Search, with a slight advantage for Nested Monte Carlo Search while experimenting on a playout heuristic. We also show how the search algorithms are bounded by the quality of the policy network, in order to improve our results the next step is to improve the policy network.

KEYWORDS
MCTS, Monte Carlo Tree Search, retrosynthesis, search algorithm

1 | INTRODUCTION

Retrosynthesis is a domain of organic chemistry that consists of finding a synthetic route (a sequence of reactions) for a given molecule in order to synthesize it from a given set of available precursor molecules [1]. It is an important part of organic chemistry molecule synthesis, and can be used to produce newfound drugs. What we aim for in this paper is to evaluate the strengths and weaknesses of two search algorithms by comparing them to AiZynthFinder's Monte Carlo Tree Search (MCTS) on a small benchmark consisting of curated and complex molecules, covering many reactions encountered by chemists.

The second section presents the retrosynthesis problem, the third section presents the AiZynthFinder retrosynthesis tool, the fourth section describes the search algorithms we compare, the fifth section details the benchmark used to compare the search algorithms, and the sixth section gives experimental results.

2 | THE RETROSYNTHESIS PROBLEM

Before diving into the details, let's broadly present the retrosynthesis problem.

- precursors: molecules that form one or multiple product molecules when they react together. ZINC [2] is a database of precursors that are available on the market.
- reaction template: a patent predicting the product of the reaction of one or multiple molecules. USPTO is a database of reaction template patents.
- One step retrosynthesis: an important part of retrosynthesis is selecting a few promising reaction templates before applying them as MCTS moves, this step uses a neural network.

As said before: the retrosynthetic analysis of a molecule is trying to find a sequence of reactions from a given molecule that leads to available precursors. It is a decision problem.

We start from the one molecule we want to find a synthetic route for and decompose it into precursors, available in the market or not. Then we recursively decompose the precursors according to a reaction template. Each time a reaction is applied, this gives us another state of the search, composed of different molecules (as many or more). The goal is to find a sequence of reaction templates (a retrosynthetic route) that leads to a state of the search uniquely composed of molecules/precursors available on the market.

Figure 1 represents a retrosynthetic route for molecule A0 (on the right) as displayed in AiZynthFinder's UI, each green framed molecule is a precursor available in ZINC, each orange framed one is a molecule that isn't available, and each black dot is a reaction. The molecules on each column represent a state of the search space that was explored, but it does not display all states explored, only the ones in the route.

3 | AIZYNTHFINDER

AiZynthFinder [3] is a retrosynthesis tool made by AstraZeneca's research and development. It has the advantage of being open source, understandable, and well described.

AiZynthFinder uses a neural network trained on USPTO, a set containing about 18 million reaction templates from organic chemistry patents. That neural network's role is to select the best reactions given a molecule we want to synthesize, it also gives a value to each move (the prior). Due to how the program works, it's hard to do without that neural network and the priors because it would require finding another method to evaluate the reactions available for a molecule. Thus, every algorithm presented here get their possible moves/reactions from the policy neural network. In this article we use AstraZeneca's open source pre-trained network. Training a network to predict more accurately the reactions for molecule retrosynthesis is another domain called "one step retrosynthesis", see Ref. [1], it is not what we aim to explore here.

AiZynthFinder takes the SMILES: a string representation of a molecule, as an input which makes the first state (which is made of only one molecule). A state is a set of molecules, from each state the neural network proposes some reactions producing a molecule from the state from precursors. If a reaction is played the molecule is removed from the state, and the precursors are added (a reaction can also be a modification of the molecule's shape only, not removing any atom, we call these "structural moves"). Retrosynthesis often uses and/or trees, here the "and" are combined into a single state as it makes the search more simple.

We use the ZINC [2] molecule database, a curated collection of commercially available (in stock) chemical compounds prepared especially for virtual screening. Any state is evaluated by AiZynthFinder's base evaluation function which is 0.95*(*fraction_of_molecules_making_the_state_in_stock*) + 0.05*(*squash _ depth_of_the_reaction_tree*). This function tries to maximize the fraction of molecules in stock among the ones composing the state. It decides between those that have the same proportion by adding the squashed (applying 1 − sigmoid of) depth of the search tree to favor shorter routes.

We don't modify it directly in this study but exploring different score functions could be interesting in future works.

4 | ALGORITHMS

Our algorithms use common primitives:

- M_{state} represents the moves available from state *state.*
- play(*state*, *m*) executes the move *m* on the state *state* and returns a children state of *state.*
- score(*state*) evaluates a state with a float value.
- *visits*(*state*) returns the number of time the state *state* has been visited.

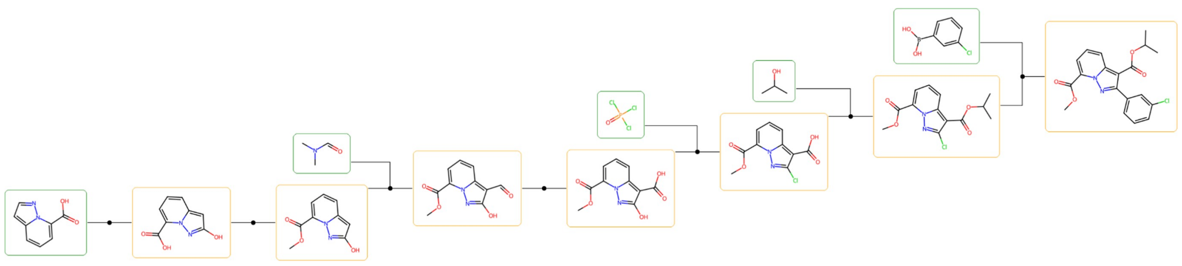

FIGURE 1 Retrosynthetic route for molecule A0.

- *sumEvals*(*state*) returns the sum of all scores returned by *state* and its child states.
- *prior*(*state*,*m*) returns the value of move *m* when applied to *state* according to the neural network (here *m* is a chemical reaction patent, and *state* a set of molecules)
- *terminal*(*state*) returns true if no move can be applied: a certain depth is reached, or we know no reaction for molecules of the *state*

4.1 | AizynthFinder's MCTS

AiZynthFinder uses a MCTS algorithm with priors very similar to PUCT. PUCT stands for "Prior Upper Confidence bounds applied to Trees", it is a generalisation of the UCT algorithm [4] using priors for each state of the problem (the prior is the policy at the output of the neural network here), see Ref. [5] for the original version of PUCT. PUCT has been used in AlphaGo [6] and Alpha Zero [7]. Just like PUCT, this MCTS algorithm explores the tree using playouts: it selects the next moves to try according to their evaluations. It plays the selected moves until it reaches a state not explored yet or until it reaches a terminal state. That state is memorized in an entry that contains the number of visits for that state, the number of visits for each child state, and the evaluations for each child state. Then the score of that newly explored state is retro-propagated to update the evaluations of the parent states.

AiZynthFinder's MCTS in Algorithm 1 differs from standard PUCT in how the *bandit* value, the value used in the selection phase of a MCTS, is determined. In all our experiments the *c* hyper-parameter is AstraZeneca's base one of 1.4. Algorithm 1 is iteratively called until it reaches the maximum number of iterations, the transposition table is conserved between iterations.

When AiZynthFinder's creators compared it to ASKCOS, the MIT and DARPA's retrosynthetic solver [8], it showed similar performances [3].

4.2 | Nested Monte Carlo Search

Monte Carlo Tree Search (MCTS) has been successfully applied to many games and problems [9].

Nested Monte Carlo Search (NMCS) [10] is an algorithm that works well for puzzles and optimization problems. It biases its playouts using lower level playouts. Online learning of playout strategies combined with NMCS has given good results on optimization problems

Algorithm 1 The MCTS algorithm
```
1: function MCTS($state$)
2:     if terminal($state$) then return score($state$)
3:     end if
4:     if $state$ is not in the transposition table then
5:         add $state$ to the transposition table
6:         return score($state$)
7:     else
8:         $best\text{-}score \leftarrow -\infty$
9:         $mean \leftarrow \frac{sumEvals(state)+prior(state,m)}{visits(state)+1}$
10:        for each $m$ in $M_{state}$ do
11:            $\mu \leftarrow mean$
12:            if $visits(play(state,m)) > 0$ then
13:                $\mu \leftarrow \frac{sumEvals(play(state,m))+prior(state,m)}{visits(play(state,m))+1}$
14:            end if
15:            $bandit \leftarrow \mu + c \times \sqrt{\frac{2\times(visits(state)+1)}{(visits(play(state,m))+1)}}$
16:            if $bandit > best\text{-}score$ then
17:                $best\text{-}score \leftarrow bandit$
18:                $best\text{-}move \leftarrow m$
19:            end if
20:        end for
21:        $res \leftarrow$ MCTS($play(state, best\text{-}move)$)
22:        update the transposition table
23:        return $res$
24:    end if
25: end function
```

[11]. Other applications of NMCS include Single Player General Game Playing [12], Cooperative Pathfinding [13], Software testing [14], heuristic Model-Checking [15], the Pancake problem [16], Games [17] and the RNA inverse folding problem [18].

Online learning of a playout policy in the context of nested searches has been further developed for puzzles and optimization with Nested Rollout Policy Adaptation (NRPA) [19]. NRPA has found new world records in Morpion Solitaire and crossword puzzles. NRPA has been applied to multiple problems: the Traveling Salesman with Time Windows problem [20, 21], 3D Packing with Object Orientation [22], the physical traveling salesman problem [23], the Multiple Sequence Alignment problem [24] or Logistics [25]. The principle of NRPA is to adapt the playout policy so as to learn the best sequence of moves found so far at each level.

The use of Gibbs sampling in Monte Carlo Tree Search dates back to the general game player Cadia Player and its MAST playout policy [26].

NMCS [10] recursively calls lower level NMCS on children states of the current state in order to decide which move to play next, the lowest level of NMCS being a random playout, selecting uniformly the move to execute among the possible moves. A heuristic can be added to the playout choices.

We detail the NMCS in Algorithm 2.

Here we used a heuristic to penalize the structural moves (when nothing is added or removed from the molecule, it only changes shape) because these moves often occupied most of the limited depth of search, even looping to a previous state sometimes. We use the score of the children (between 0 and 1), to which we add 1 if the largest molecule weight in the state is smaller than its parent largest molecule weight. That value is then used as the chance to select that move over the sum of every other move's values. The modified score function for the heuristic and the heuristic are described in Algorithm 3.

While we did not use softmax to harden the heuristic, nor tuned the parameters, that simple heuristic allowed us to diminish the structural moves problem, giving them less than half the chance of being selected in playouts than before, and led to better results.

4.3 | Greedy Best First Search

GBFS stands for Greedy Best-First Search. It is a simple algorithm that consists of opening (and removing) the best node from a list of nodes sorted by their scores, evaluating all its children and inserting them in the sorted list of nodes, and then repeating the operation by opening the new best node [27]. The evaluation function can use playouts to make the algorithm closer to a Monte Carlo Search algorithm. Like MCTS, that algorithm can lock itself in a local minimum, but is faster (and less accurate) as it skips the playout and the associated calculations between each node discovery. Both lack forced progress in depth of NMCS. We describe GBFS in Algorithm 4.

The function insert(*open-states, score, new-state*) inserts *new-state* in the sorted list *open-states* given the *score* value.

We modified the evaluation function similarly to the NMCS playout function: if the children's biggest molecule is smaller than the parent's then add 1 to the score. That modification allows to avoid structural moves, multiplying states with high scores, but also prevents structural moves that are sometimes necessary to the resolution of a molecule, finding an alternative solution would greatly help this algorithm.

5 | BENCHMARK

To compare our algorithms to AstraZeneca's MCTS, we use a small subset containing 40 SMILES representation of drugs from the PubChem database (see Table 3) selected by Ref. [28] (molecules ID starting with C) and 20 SMILES representation of molecules selected by Bayer's chemists [29] (molecules ID starting with A). The Bayer's chemists molecules contains molecules ranging from easy ones to hard ones. The 40 molecules selected randomly from PubChem by the authors of the benchmark were obtained in such a way to cover small to large molecules [28]. The original goal of this benchmark was to test the prediction of difficulty of retrosynthesis of these molecules, thus these 40 molecules feature some of the hardest to synthesize according to some chemists.

Generally, one step retrosynthesis uses USPTO-50 K [30] as a benchmark. However here we are not trying to benchmark the reaction propositions of the neural network used here, but how our algorithm solves the retrosynthesis problem. Thus our benchmark provides a few advantages: proposing harder molecules to synthesize, reducing the benchmark size allowing us to focus more on each molecule, and giving a fixed benchmark to compare with others.

You can find the SMILES representation of this benchmark in Table 3 of the appendix.

Algorithm 2 The NMCS algorithm.

```
1:  function NMCS($c\text{-}st$, $lv$)
2:      if $lv = 0$ then                                    ▹ Playout if lv = 0
3:          $ply \leftarrow 0$
4:          $seq \leftarrow \{\}$
5:          while not terminal($c\text{-}st$) do
6:              $move \leftarrow$ heuriChoice($M_{c\text{-}st}$)
7:              $c\text{-}st \leftarrow play(c\text{-}st, move)$
8:              $seq[ply] \leftarrow move$
9:              $ply \leftarrow ply + 1$
10:         end while
11:         return (score ($c\text{-}st$), $seq$)
12:     else
13:         $best\text{-}score \leftarrow -\infty$
14:         $best\text{-}sequence \leftarrow [\,]$
15:         $ply \leftarrow 0$
16:         while $c\text{-}st$ is not terminal do
17:             for each $move$ in $M_{c\text{-}st}$ do
18:                 $n\text{-}st \leftarrow play(c\text{-}st, move)$
19:                 $(score, seq) \leftarrow$NMCS($n\text{-}st, level - 1$)
20:                 if $score \geq best\text{-}score$ then
21:                     $best\text{-}score \leftarrow score$
22:                     $best\text{-}sequence[ply..] \leftarrow move + seq$
23:                 end if
24:             end for
25:             $next\text{-}move \leftarrow best\text{-}sequence[ply]$
26:             $ply \leftarrow ply + 1$
27:             $c\text{-}st \leftarrow play(c\text{-}st, next\text{-}move)$
28:         end while
29:         return ($best\text{-}score, best\text{-}sequence$)
30:     end if
31: end function
```

6 | RESULTS

These experiments were made on a 3.50GHz intel core i5-6600 K on Windows 10 with 32 Gb of RAM. We used the same common parameters for every algorithm:

- Max step for substrates (how many reactions we can make from a substrate to the target molecule): 15
- Policy cutoff cumulative: 0.995
- Policy cutoff number (maximum number of possible moves returned on a molecule): 50
- Filter cutoff: 0.05

6.1 | AiZynthFinder's MCTS

To compare our algorithms, we ran AiZynthFinder MCTS at least 2 times on each of the 60 molecules of the benchmark with the base settings, C = 1.4 for the exploration/exploitation constant. Running it a few times only is not problematic because the MCTS results were observed to be very stable on the few molecules we ran it multiple times on. Our goal is not to measure the exact solving times as they heavily depend on the implementation, the language, and the hardware, but to see how many molecules of the benchmark a given algorithm can find a synthetic route for. The times specified are here only to give an idea of the differences in performance between the algorithms.

First, we ran the MCTS (Table 1) with a timeout of 2 minutes and a maximum number of iterations of 100 ("MCTS 2 min 100it"). The molecules identified with a C in Table 3 were either solved instantly (<200 ms) or not solved in 2 minutes. On the contrary, the molecules selected by Bayer's chemists (starting with "A") took generally more time to be solved if they were. In addition, a bigger proportion of molecules from A were solved than from C, which can be explained by their sizes and the presence of distinctive atoms (like fluor). We then

Algorithm 3 The modified score function and heuristic choice.

```
1:  function val($s1$, $s2$)
2:      if biggest molecule from $s1$ is smaller than biggest molecule from $s2$ then
3:          return $score(s1) + 1$
4:      end if
5:      return $score(s1)$
6:  end function
7:  function HeuriChoice($moves$)
8:      $weights \leftarrow$ [val($play(st, m)$) for $m$ in $M_{st}$]
9:      $su \leftarrow 0$
10:     $rd \leftarrow randomRange(0, sum(weights))$
11:     $ind \leftarrow 0$
12:     for $w$ in $weights$ do
13:         $su \leftarrow su + w$
14:         if $su \geq rd$ then return $ind$
15:         end if
16:         $ind \leftarrow ind + 1$
17:     end for
18:     return $ind$
19: end function
```

Algorithm 4 The GBFS algorithm.

```
1:  function GBFS($ini\text{-}state$, $max\text{-}iter$)
2:      $open\text{-}states \leftarrow [ini\text{-}state]$
3:      $state \leftarrow ini\text{-}state$
4:      $iter \leftarrow 0$
5:      $best\text{-}state \leftarrow ini\text{-}state$
6:      $best\text{-}score \leftarrow$ score($ini\text{-}state$)
7:      while not optimal($state$) and $open\text{-}states \neq []$ and $iter < max\text{-}iter$ do
8:          $iter \leftarrow iter + 1$
9:          $state \leftarrow$ pop( $open\text{-}states$, 0)
10:         for each $move$ in $M_{state}$ do
11:             $new\text{-}state \leftarrow$ play($state$, $move$)
12:             $score \leftarrow$ score($new\text{-}state$)
13:             insert($open\text{-}states$, $score$, $new\text{-}state$)
14:             if $score \geq best\text{-}score$ then
15:                 $best\text{-}state \leftarrow state$
16:                 $best\text{-}score \leftarrow score$
17:             end if
18:         end for
19:     end while
20:     return $best\text{-}state$
21: end function
```

launched the MCTS with a time limit of 20 minutes, solving a few more molecules, and finally, we launched a MCTS of 2 h on some of the remaining ones: C2, C35, C37, C38. Only C37 (*) got solved in 5236.093 seconds, it was not included in Figure 2.

As you can see on Table 1, increasing the MCTS search time doesn't help much, the molecules are either solved instantaneously (<200 ms) or very quickly. The instantaneous solving is due to the neural network (one-step retrosynthesis) which immediately proposes the right solution in 1 or 2 reactions for small molecules. This means that the quality of the search algorithm doesn't matter on these molecules, and is solved almost equally as fast with the MCTS, the NMCS, the GBFS, and even a NMCS without playout. These molecules are

TABLE 1 AiZynthFinder's MCTS results.

ID	Time (s)	ID	Time (s)	ID	Time (s)	ID	Time (s)
C14	0.060	C13	0.061	C21	0.061	C31	0.063
C34	0.063	C25	0.064	C40	0.066	C26	0.071
C5	0.080	A11	0.120	C8	0.125	C3	0.130
C23	0.149	C1	0.167	A19	0.343	A9	0.743
A14	0.792	A4	0.821	A17	2.564	A1	5.225
A16	12.490	A2	12.948	A18	15.447	C20	22.088
C36	41.294	A7	84.585	C19	310.966	C22	425.647
A15	587.830	A5	1086.547	C2	>7200	C4	>7200
C6	>7200	C7	>7200	C9	>7200	C10	>7200
C11	>7200	C12	>7200	C15	>7200	C16	>7200
C17	>7200	C18	>7200	C24	>7200	C27	>7200
C28	>7200	C29	>7200	C30	>7200	C32	>7200
C33	>7200	C35	>7200	C37*	>7200	C38	>7200
C39	>7200	A0	>7200	A3	>7200	A6	>7200
A8	>7200	A10	>7200	A12	>7200	A13	>7200

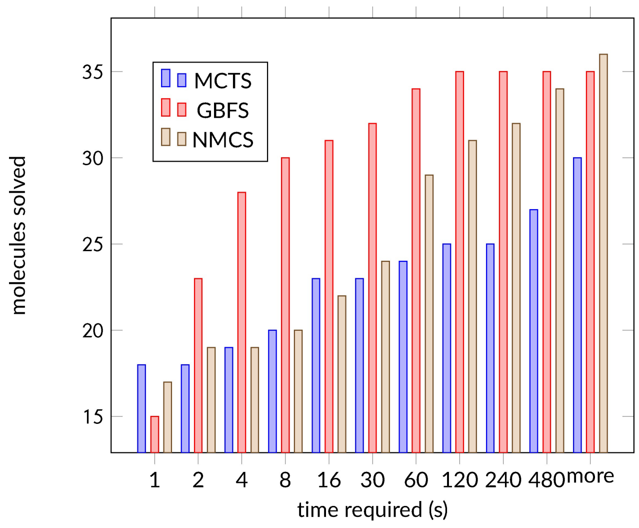

FIGURE 2 Distribution of the numbers of molecules solved with times in seconds.

not useful to our research so we remove them from the set for further experiments.

6.2 | Other algorithms

Every molecule solved by the MCTS was also solved by NMCS and all but one by GBFS. Even if they may be slower than MCTS for easy states (the GBFS has to instance 50 children per opened state even if it uses only one or two). Thus, we are going to focus on molecules not solved by AizynthFinder base MCTS and those that took at least 2 minutes to solve.

For the NMCS, we first perform a level 1 NMCS using only the 5 best moves from each state, instead of the 50 best given by the Policy cutoff number. If that NMCS (usually shorter than 1 minute) fails we perform a much longer level 1 NMCS using all the 50 moves. If even that fails, we launch the level 2 NMCS. The level 2 NMCS is very slow but is able to solve molecules unsolved by both MCTS and GBFS.

The GBFS was only launched once on each molecule because the algorithm is deterministic, it was launched with a time limit of 20 minutes, and molecules not solved by then are considered unsolved. Given enough time, the GBFS explores the entire tree.

First, we can notice in Figure 2 that NMCS and GBFS outperform Astrazeneca's MCTS in the long run, but as the y-axis starts at 15, they slightly underperform on molecules solved by 1 or 2 steps, in less than 1 s. These molecules take about 0.6 s with GBFS and NMCS compared to about 0.1 s for MCTS. This is because they don't open first the most promising node according to the neural network. It is observed that GBFS is better than both NMCS and MCTS for search times between 2 and 480 seconds. It stops improving after 120 seconds while the others continue to improve. NMCS finds retrosynthesis routes slower but can find more of them. MCTS (or any algorithm opening the reaction greedily according to the neural network) is better for very short experiments ($<1\,s$), GBFS is better for medium length experiments ($<60s$) and NMCS appears to be better for longer experiments and more complex molecules.

On Table 2 we focus our attention on the molecules that took more than 2 minutes to solve or were not solved by at least one of the algorithms:

C19, C20, C22, C37, A0, A3, A5, A6, A8, A12, A13 and A15

The results on A8 (*) were obtained by opening the 200 best nodes given by the neural network and not the 50 best, in addition to only searching up to a depth of 5 instead of 15. NMCS and MCTS were unable to solve the molecule with the same parameters. The reactions required to solve A8 were not present in the 50 best proposals from the neural network, but in the 200 best. On the other hand, a top 5 level 1 NMCS was enough to solve 30 of the 60 molecules, meaning the NN was very accurate in these cases. It emphasizes how much results depend on the accuracy of the NN. Again, the one we used was trained by AiZynthFinder's team on the public USPTO reaction dataset, which does not feature many reactions present in licensed datasets such as Reaxys or Pistachio [31].

Our GBFS was unable to solve C20, despite being solved by the MCTS and the NMCS, we think it was because our search heuristic favors the non structural moves when a structural move is required here to cut the carbon cycle. Overall, molecules with long carbon cycles posed problems to be solved to all the algorithms and C20 was the smallest and most simple of them from the benchmark. It is what AstraZeneca is focusing on in the latest updates for AiZynthFinder (we used an earlier version from 2022 to conduct these experiments).

Like with MCTS, running the other algorithms longer did not yield much improvement. This is because the neural network does not always propose the best reactions. Obtaining a better network, possibly trained on a more complete dataset could improve our results greatly.

TABLE 2 Comparison of algorithms.

Molecule	MCTS	GBFS	NMCS
C19	310.966	3.765	81.267
C20	119.230	X	46.166
C22	425.647	50.800	4.255
C37	5236.093	2.836	8.919
A0	X	4.569	84.582
A3	X	2.075	176.445
A5	1086.547	2.145	60.000
A6	X	29.604	1075.222
A8	X	95.365*	X
A12	X	47.78	431.095
A13	X	X	518.727
A15	587.830	3.587	37.178

To put our results in perspective, out of the 60 molecules of our benchmark we managed to solve 35 molecules with GBFS, and 36 with NMCS [29], while managed to solve 38 molecules with a MCTS and 41 with a DFPN (Depth-First Proof Number search, AizynthFinder's DFPN does not yield such results). We hope we will be able to try with a more complete dataset and the according NN in the future. Bigger molecules would still be a challenge given all the reactions and subtrees they offer, but we think it could help with the few unsolved small molecules: C4, C6, C7, C10, C12, C35, C38, and A10.

7 | CONCLUSION

While MCTS solves 31 molecules out of 60 from this benchmark, GBFS solves 35 in a reasonable time and NMCS solves 36. We showed that GBFS and NMCS could provide satisfying performance improvements, especially since GBFS and NMCS are much simpler and don't use the neural network as a search policy beyond the reaction proposition, unlike MCTS. We believe that a more accurate neural network trained on a bigger dataset, and a more complete template set would improve the performances.

8 | FUTURE WORKS

Retrosynthesis is a vast topic, and much remains to be done, we only scratched the surface of AiZynthFinder here. It would be interesting to experiment on more algorithms, including the canonical PUCT and apply the prior to the algorithms we used here, use other score functions or train and use another neural network. This research would require a lot of time, and we can only encourage other computer scientists to try their algorithms and score functions on AiZynthFinder.

9 | APPENDIX

The appendix can be found in Table 3 below.

ACKNOWLEDGMENTS

This work was supported in part by the French government under the management of Agence Nationale de la Recherche as part of the "Investissements d'avenir" program, reference ANR19-P3IA-0001 (PRAIRIE 3IA Institute).

TABLE 3 Benchmark

Mol	SMILES
C1	COc4ccc3nc(NC(=O)CSc2nnc(c1ccccc1C)[nH]2)sc3c4
C2	OC8Cc7c(O)c(C2C(O)C(c1ccc(O)c(O)c1)Oc3cc(O)ccc23)c(O) c(C5C(O)C(c4ccc(O)c(O)c4)Oc6cc(O)ccc56)c7OC8c9ccc(O)c(O)c9
C3	NC(=O)Nc1nsnc1C(=O)Nc2ccccc2
C4	C=CCn5c(=O)[nH]c(=O)c(=C4CC(c2ccc1OCOc1c2)N(c3ccccc3)N4)c5=O
C5	Oc1c(Cl)cc(Cl)cc1CNc2cccc3cn[nH]c23
C6	CC(C)C(C)C=CC(C)C1CCC3C1(C)CCC4C2(C)CCC(O)CC25CCC34OO5
C7	CC45CC(O)C1C(CC=C2CC3(CCC12)OCCO3)C4CCC56OCCO6
C8	CCc2ccc(c1ccccc1)cc2
C9	CC5C4C(CC3C2CC=C1CC(OC(C)=O)C(O)C(O)C1(C)C2CC(O)C34C)OC56CCC(=C)CO6
C10	CSc2ncnc3cn(C1OC(CO)C(O)C1O)nc23
C11	CCc1c(C)c2cc5nc(nc4[nH]c(cc3nc(cc1[nH]2)C(=O)C3(C)CC)c(CCC(=O)OC)c4C)C(C)(O)C5(O)CCC(=O)OC
C12	CN(COC(C)=O)c1nc(N(C)COC(C)=O)nc(N(C)COC(C)=O)n1
C13	CSc2ccc(OCC(=O)Nc1ccc(C(C)C)cc1)cc2
C14	Cc2ccc(C(=O)Nc1ccccc1)cc2
C15	CC5CC(C)C(O)(CC4CC3OC2(CCC1(OC(C=CCCC(O)=O)CC=C1)O2)C(C)CC3O4)OC5C(Br)=C
C16	COc8ccc(C27C(CC1C5C(CC=C1C2c3cc(OC)ccc3O)C(=O) N(c4cccc(C(O)=O)c4)C5=O)C(=O)N(Nc6ccc(Cl)cc6Cl)C7=O)cc8
C17	CC=CC(O)CC=CCC(C)C(O)CC(=O)NCC(O)C(C)C(=O)NCCCC2OC1(CCCC(CCC(CC=C(C)C(C)O)O1)CCC2C
C18	CCC(C)=CC(=O)OC1C(C)CC3OC1(O)C(O)C2(C)CCC(O2)C(C)(C)C=CC(C)C3=O
C19	CCC(CO)NC(=O)c2cccc(S(=O)(=O)N1CCCCCC1)c2
C20	CCCCCC1OC(=O)CCCCCCCCC=CC1=O
C21	COc1ccc(Cl)cc1
C22	CC(C)(C)C(Br)C(=O)NC(C)(C)C1CCC(C)(NC(=O)C(Br)C(C)(C)C)CC1
C23	COc2cc(CNc1ccccc1)ccc2OCC(=O)Nc3ccc(Cl)cc3
C24	COC4C=C(C)CC(C=CC=CC#CC1CC1Cl)OC(=O)CC3(O)CC(OC2OC(C)C(O)C(C)(O)C2OC)C(C)C(O3)C4C
C25	CCc2ccc(OC(=O)c1ccccc1Cl)cc2
C26	COc1ccccc1c2ccccc2
C27	CCCC(NC(=O)C1CC2CN1C(=O)C(C(C)(C)C)NC(=O)Cc3cccc (OCCCO2)c3)C(=O)C(=O)NCC(=O)NC(C(O)=O)c4ccc(NS-(N)(=O)=O)cc4
C28	COC4C(O)C(C)OC(OCC3C=CC=CC(=O)C(C)CC(C) C(OC2OC(C)CC1(OC(=O)OC1C)C2O)C(C)C=CC(=O)OC3C)C4OC
C29	CC(C)(C)c4ccc(C(=O)Nc3nc2C(CC(=O)NCC#C)C1(C)CCC(O)C(C)(CO)C1Cc2s3)cc4
C30	CCC7(C4OC(C3OC2(COC(c1ccc(OC)cc1)O2)C(C)CC3C)CC4C) CCC(C6(C)CCC5(CC(OCC=C)C(C)C(C(C)C(OC)C(C)C(O)=O)-O5)O6)O7
C31	O=C(OCc1ccccc1)c2ccccc2
C32	CC(C)CC(NC(=O)C(CC(=O)NC2OC(CO)C(OC1OC(CO)C(O)C(O) C1NC(C)=O)C(O)C2NC(C)=O)NC(=O)c3ccccc3)C(=O)NC(C(C)O)C(N)=O
C33	CCCC5OC(=O)C(C)C(=O)C(C)C(OC1OC(C)CC(N(C)C)C1O)C(C) (OCC=Cc3cnc2ccc(OC)cc2c3)CC(C)C4=NCCN6C(C4C)-C5(C)OC6=O
C34	COC(=O)c1ccccc1NC(=O)CC(c2ccccc2)c3ccccc3
C35	Cc4onc5c1ncc(Cl)cc1n(C3CCCC(CNC(=O)OCc2ccccc2)C3)c(=O)c45
C36	CC(C)OCCCNC(=O)c3cc2c(=O)n(C)c1ccccc1c2n3C
C37	COC(=O)N4CCCC(N3CCC(n1c(=O)n(S(C)(=O)=O)c2ccccc12)CC3)C4
C38	Cc5c(C=NN3C(=O)C2C1CC(C=C1)C2C3=O)c4ccccc4n5Cc6ccc(N(=O)=O)cc6
C39	CCC5OC(=O)C(C)C(=O)C(C)C(OC1OC(C)CC(N(C)C)C1O)C(C) (OCC#Cc4cc(c3ccc2ccccc2n3)no4)CC(C)C(=O)C(C)C6NC-(=O)OC56C
C40	CC(=O)Nc1ccccc1NC(=O)COc2ccccc2
A0	COC(=O)c1cccc2c(C(=O)OC(C)C)c(nn12)c3cccc(Cl)c3

TABLE 3 (Continued)

Mol	SMILES
A1	CCOC(=O)C1=C(C)N(C)C(=O)NC1c2ccncc2
A2	Cn1c(nc2ccccc12)c3ncc(cc3N4CCCC4=O)c5ccccc5
A3	CC1(COc2ccccc2)CCN1C(=O)c3ccccc3
A4	Cc1cc(nn1CC(=O)N2CCC(CC2)c3nc(cs3)C4=NOC(C4)c5ccccc5OCC#C)C(F)(F)F
A5	CCC1(CC1)c2ccc(C)cc2CN(C3CC3)C(=O)c4c(F)n(C)nc4C(F)F
A6	CC1CCC(CN1C(=O)c2ccnc(NS(=O)(=O)c3cccc(Cl)c3)n2)c4cncc(c4)c5cnn(c5)C(=O)C
A7	Cc1cccc(C)c1c2csc(n2)C(=O)NCCC3=CC(=CC(=O)N3)Oc4ccccc4Cl
A8	COc1ccc(cc1)N2CC(CC2C(=O)NCc3ccc4CCCCc4n3)NCC(F)(F)F
A9	FC(F)(F)Oc1cc(Cl)cc(c1)n2cnc3ccc(cc23)S(=O)(=O)NC4COC4
A10	COc1cc2c3CC(NC(=O)C)C(Oc3ccc2cc1C#N)c4ccc(F)c(F)c4
A11	FC(F)(F)c1ccc(Nc2ncc(C(=O)NCC3CCC(F)(F)CC3)c(n2)C(F)(F)F)c(Cl)c1
A12	Fc1cnc(Nc2ccc(cc2)C(=O)N3CCN(CC3)C4COC4)nc1c5cnc(n5C6CCCCC6)C(F)(F)F
A13	Cc1ccc(C2=NC(O)C(=O)Nc3cc(C)c(Cl)cc23)c(Cl)c1
A14	CC1(C)CN(C(C(=O)NC2CCCCC2)c3cccc(c3)C(F)(F)F)C(=O)C1
A15	CC(=O)Nc1ccc(cc1)S(=O)(=O)c2ccc(cc2)C3CCN(C3)c4ccccc4
A16	Cc1cnc(C(=O)O)c(OC(F)F)c1
A17	FC(F)(F)c1nnc2CNCCn12
A18	CNCCC(Oc1cccc2ncncc12)c3cncs3
A19	FC(F)(F)c1nnc2CN(CCn12)c3cccc(I)c3

DATA AVAILABILITY STATEMENT

The data that supports the findings of this study are available in the supplementary material of this article

ORCID

Milo Roucairol http://orcid.org/0000-0002-7794-5614

REFERENCES

1. M. H. Lin, Z. Tu, C. W. Coley, *J Cheminform* **2022**, *14*, 15.
2. J. J. Irwin, B. K. Shoichet, *J. Chem. Inf. Model.* **2005**, *45*, 177–182.
3. S. Genheden, A. Thakkar, V. Chadimová, J. L. Reymond, O. Engkvist, E. Bjerrum, *J Cheminform* **2020**, *12*, 70..
4. L. Kocsis, C. Szepesvári. In: J. Fürnkranz, Scheffer T, M. Spiliopoulou, editors. Bandit Based Monte-Carlo Planning, vol. 4212 of Lecture Notes in Computer Science Berlin, Heidelberg: Springer Berlin Heidelberg, **2006**, p. 282–293. http://link.springer.com/10.1007/11871842 29.
5. C. D. Rosin, *Ann Math Artif Intell* **2011**, *61*, 203–230.
6. D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G v. d. Driessche, et al., *Nature* **2016**, *529*, 484–489.
7. D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, et al., *Science* **2018**, *362*, 1140–1144.
8. C. W. Coley, R. Barzilay, T. S. Jaakkola, W. H. Green, K. F. Jensen, *ACS Central Science* **2017**, *3*, 434–443.
9. C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, et al., A Survey of Monte Carlo Tree Search Methods. IEEE Transactions on Computational Intelligence and AI in Games, **2012**, *4*, 1–43.
10. T. Cazenave. Nested Monte-Carlo Search. In: IJCAI, **2009**, p. 456–461.
11. A. Rimmel, F. Teytaud, T. Cazenave, Optimization of the Nested Monte-Carlo Algorithm on the Traveling Salesman Problem with Time Windows. In: EvoApplications, vol. 6625 of LNCS Springer, **2011**. p. 501–510.
12. J. Méhat, T. Cazenave, Combining UCT and Nested Monte Carlo Search for Single-Player General Game Playing. IEEE Transactions on Computational Intelligence and AI in Games, **2010**, *2*, 271–277.
13. B. Bouzy, Monte-Carlo Fork Search for Cooperative Path-Finding. In: Computer Games Workshop at IJCAI, **2013**, p. 1–15.
14. S. M. Poulding, R. Feldt, Generating structured test data with specific properties using nested Monte-Carlo search. In: GECCO, **2014**. p. 1279–1286.
15. S. M. Poulding, R. Feldt, Heuristic Model Checking using a Monte-Carlo Tree Search Algorithm. In: GECCO, **2015**, p. 1359–1366.
16. B. Bouzy, Burnt Pancake Problem: New Lower Bounds on the Diameter and New Experimental Optimality Ratios. In: SOCS, **2016**, p. 119–120.
17. T. Cazenave, A. Saffidine, M. J. Schofield, M. Thielscher, Nested Monte Carlo Search for Two-Player Games. In: AAAI, **2016**, p. 687–693.
18. F. Portela, An unexpectedly effective Monte Carlo technique for the RNA inverse folding problem. BioRxiv, **2018**, p. 345587.
19. C. D. Rosin, Nested Rollout Policy Adaptation for Monte Carlo Tree Search. In: IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, **2011**, p. 649–654.

20. T. Cazenave, F. Teytaud, Application of the Nested Rollout Policy Adaptation Algorithm to the Traveling Salesman Problem with Time Windows. In: Learning and Intelligent Optimization - 6th International Conference, LION 6, **2012**, p. 42–54.
21. S. Edelkamp, M. Gath, T. Cazenave, F. Teytaud, Algorithm and knowledge engineering for the TSPTW problem. In: Computational Intelligence in Scheduling (SCIS), 2013 IEEE Symposium on IEEE, **2013**, p. 44–51.
22. S. Edelkamp, M. Gath, M. Rohde, Monte-Carlo Tree Search for 3D Packing with Object Orientation. In: KI 2014: Advances in Artificial Intelligence Springer International Publishing, **2014**, p. 285–296.
23. S. Edelkamp, C. Greulich, Solving physical traveling salesman problems with policy adaptation. In: Computational Intelligence and Games (CIG), 2014 IEEE Conference on IEEE, **2014**, p. 1–8.
24. S. Edelkamp, Z. Tang, Monte-Carlo Tree Search for the Multiple Sequence Alignment Problem. In: Proceedings of the Eighth Annual Symposium on Combinatorial Search, SOCS 2015 AAAI Press, **2015**, p. 9–17.
25. S. Edelkamp, M. Gath, C. Greulich, M. Humann, O. Herzog, M. Lawo, Monte-Carlo Tree Search for Logistics. In: Commercial Transport Springer International Publishing, **2016**, p. 427–440.
26. H. Finnsson, Y. Björnsson, Simulation-Based Approach to General Game Playing. In: Aaai, vol. 8, **2008**, p. 259–264.
27. J. E. Doran, D. Michie, Experiments with the graph traverser program. Proceedings of the Royal Society of London Series A Mathematical and Physical Sciences, **1966**, *294*, 235–259.
28. P. Ertl, A. Schuffenhauer, *J Cheminform* **2009**, *1*, 8.
29. C. Franz, G. Mogk, T. Mrziglod, K. Schewior, Completeness and Diversity in Depth-First Proof-Number Search with Applications to Retrosynthesis. In: Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence Vienna, Austria: International Joint Conferences on Artificial Intelligence Organization, **2022**, p. 4747–4753. https://www.ijcai.org/proceedings/2022/658.
30. D. M. Lowe, Extraction of chemical structures and reactions from the literature. Thesis, University of Cambridge, **2012**.
31. A. Thakkar, T. Kogej, J. L. Reymond, O. Engkvist, E. J. Bjerrum, *Chem. Sci.* **2020**, *11*, 154–168.

How to cite this article: M. Roucairol, T. Cazenave, *Molecular Informatics* **2024**, *43*, e202300259. https://doi.org/10.1002/minf.202300259

Graphical Abstract

The contents of this page will be used as part of the graphical abstract of html only.
It will not be published as part of main.