Monte-Carlo tree search applied to structure generation

Milo Roucairol

PSL
UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à Dauphine

Structure generation using Monte Carlo Search

Soutenue par
Milo ROUCAIROL
Le 13 12 2024

École doctorale n°543
SDOSE

Spécialité
Informatique

Composition du jury :

Alexandre VARNEK Professeur, Université de Strasbourg	*Président*
Carola DOERR Directrice de recherche, CNRS, Sorbonne Université	*Rapporteur*
Nicolas JOUANDEAU Professeur des universités, Université Paris 8	*Rapporteur*
Tristan CAZENAVE Professeur, LAMSADE - Université Paris-Dauphine - PSL	*Directeur de thèse*

PSL

Contents

Chapter 1

Introduction

1.1 Structure

You probably already have an idea of what is a structure. It is a word frequently used but we noticed many would struggle to give it a formal definition.

Here is the definition of structure we use here: "system defined by its fundamental elements, and the roles between these elements".

Intuitively, many things can be described as structures: buildings, images, programs, society, living beings, knowledge, and matter. It is hard to find something that has no structure or cannot be thought of as a structure, and even harder to find something that has no structure and is interesting to generate. As such, this thesis covers a wide range of themes and subjects and is a collection of applications, from molecules, coalition of agents, nonograms, or abstract graphs.

Generative Artificial Intelligence is a topic that recently saw a massive surge of papers, investments, and media coverage. First with the introduction of Generative Adversarial Networks in 2014, with the reveal of Dall-E in 2021, ChatGPT in 2022, and now (2024) with AI capable of generating entire minutes of video. These advances provide stunning results, photorealistic pictures or indiscernible from ones made by real artists, coherent sentences and paragraphs (mostly), and not too jarring videos. However these AI still present limits: ChatGPT sometimes hallucinates, the skin can be too smooth, some fingers are not always done right, and the videos lack coherence. In addition to the deep learning AI limitations and ethics considerations, these AI are not designed for and fall short on problem-solving, especially combinatorial.

The generation of structures using Monte Carlo Search algorithms have multiple advantages:

1. They are explainable.

2. They require no training, or training over less extensive datasets.
3. They perform well on strictly defined problems and always provide a safe and valid output.
4. They don't replace artists but instead help non-artistic workers.

Conversely, while they perform well in exact sciences, they are inappropriate for image and video generation.

1.2 Monte Carlo Search

To generate structures, we decided to focus exclusively on Monte Carlo Search (MCS) algorithms.

Monte Carlo Search algorithms are a class of randomized search algorithms involving some form of machine learning. They use random or guided exploration to sample the search space and guide future searches using that information. They are anytime, i.e. they can give a valid (but not optimal) solution to a problem even if they are stopped prematurely. The quality of the solution that is returned increases with the running time. They give a nonoptimal solution in a determined time, Las Vegas algorithms give an optimal solution in a non-determined time.

Monte Carlo algorithms trace back to 1953 with the Monte Carlo method from Nicholas Metropolis, mainly used in physics. That algorithm was not using machine learning yet, but inspired multiple (and very different) algorithms that came after. That was not until 2006 that the term "Monte Carlo Tree Search" was coined. (Coulom, R. (2007). Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In: van den Herik, H.J., Ciancarini, P., Donkers, H.H.L.M.(. (eds) Computers and Games. CG 2006. Lecture Notes in Computer Science, vol 4630. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-75538-8_7)

The same year, the most prominently MCTS algorithm used now, Upper Confidence Bound applied to trees (UCT), was invented. (Kocsis, L., Szepesvári, C., & Willemson, J. (2006). Improved monte-carlo search. Univ. Tartu, Estonia, Tech. Rep, 1, 1-22.)

Monte Carlo was then used as a gameplaying AI for games with large search spaces, like Go. In 2017 AlphaGo, a mixture of deep neural network and MCTS, managed to beat the world champion of Go. But has since then been used on many other problems, providing state of the art on difficult deterministic problems as you will see in this thesis. These problems include industry, chemistry, transportation networks, mathematics, graph theory, and medicine. Anything that can be explored according to a set of rules, and can be evaluated rapidly by a computer, can be optimized by Monte Carlo Search algorithms.

Monte Carlo Search algorithms are divided into two categories:

1. Iterative, also called Monte Carlo Tree Search. It maps the search space with a tree storing the previous results and use that knowledge to direct further searches.

2. Recursive, also called the nested family. Instances of the algorithm call lower level versions of the algorithm to evaluate and select the best move available.

The iterative family includes UCT and the canonical MCTS algorithm. It is the most widely used family of MCS algorithms as UCT is featured in almost all research that involves MCS. However that family of algorithms has two shortcomings: they revisit states, thus spending computing time needlessly when locked in a local minimum, and they don't optimize toward the end of the search tree. The existing recursive algorithms avoid these problems, but in return, they are able to miss states entirely and give no guarantee of exploring the entire search tree (which should never happen anyway since MCS algorithms are made for when the search space is too large to explore entirely).

1.3 Search Space And Terms

All works featured in this thesis share the same concepts and terminology. A search algorithm explores a search space, it can be continuous or discrete, here we focused on discrete graph-like search spaces.

A discrete search space is defined by the states composing it and the link between those states, the moves.

1.3.1 State

A state is a complete or incomplete form of the solution that the search algorithm is expected to return. A state can also include additional informations that are not relevant to the solution but help decide moves during the search.

We can distinguish two special types of states:

1. The initial state. The algorithms start the search from this default start state.

2. The final states. A state can be considered final when it is valid and can evaluated by a simple algorithm, or does not have any moves allowing it to turn it into another state (terminal).

1.3.2 Move

A move is the transformation of one state to another. Not all states can be transformed into any other state, the move has to be legal (allowed by

the search space). This ensure that the output solution returned by the algorithm is always valid if not optimal (for example a knight can't move in a straight line in a game of chess).

1.3.3 Playout

Search algorithms need ways to evaluate the states they encounter during their search. These evaluation functions take many forms, for games they count the score of a final state, for transportation networks they compute a set of metrics...

To be evaluated, a state needs to be final: either be able to be returned as a valid solution by the algorithm, or be terminal (if it is invalid the lowest value is often returned). But nonterminal states of the search space are not always final, it depends on the problem but for example, the HP-model has non-final intermediate states, while all Coalition Structure Generation (CSG) states are final.

When a state is not final (a direct evaluation using a simple function is not possible), Monte Carlo Search algorithms use playouts to randomly, or according to a policy, apply moves to update the state and reach a final state. The playout then evaluates the non-final state the playout originated from using the evaluation(s) of the state at the end of (or encountered during) the playout.

Recent advances also replaced the playouts with neural networks to evaluate the non-final states directly.

1.4 Introductory Example

Here is an introductory example using the graph generation problem. We try to maximize the largest eigenvalue of the adjacency matrix for trees with 10 vertices.

1. State: the adjacency matrix.
2. Starting State: empty matrix of size 10 by 10.
3. Final State: the sum of all matrix cells is equal to 20.
4. Legal Moves: turn a 0 cell into a 1 on the adjacency matrix on a column whose sum is equal to 0 (and another cell symmetrically to the diagonal).
5. Score Function (maximisation): max(eigenvalues(adjacency matrix)).
6. Playout: randomly selects a legal move until none is available, then returns the last state.

Nothing else is needed other than the adjacency matrix to represent the state. The score function and the moves do not require additional information.

The Starting state is equivalent to a one-node graph without an edge. But it leaves room for up to 9 new nodes and edges. The final state can't exceed 10 nodes and 9 edges due to the construction of the legal moves and the starting state.

The legal moves must conserve the symmetry of the adjacency matrix and modify cells in position (x,y) if it modifies a cell in position (y,x). The condition on the column sum ensures we are adding a leaf and not an edge connecting two already existing nodes.

The Score function takes the adjacency matrix (full or not), computes the eigenvalues, and returns the largest one.

The playouts are guaranteed to return a size 10 tree.

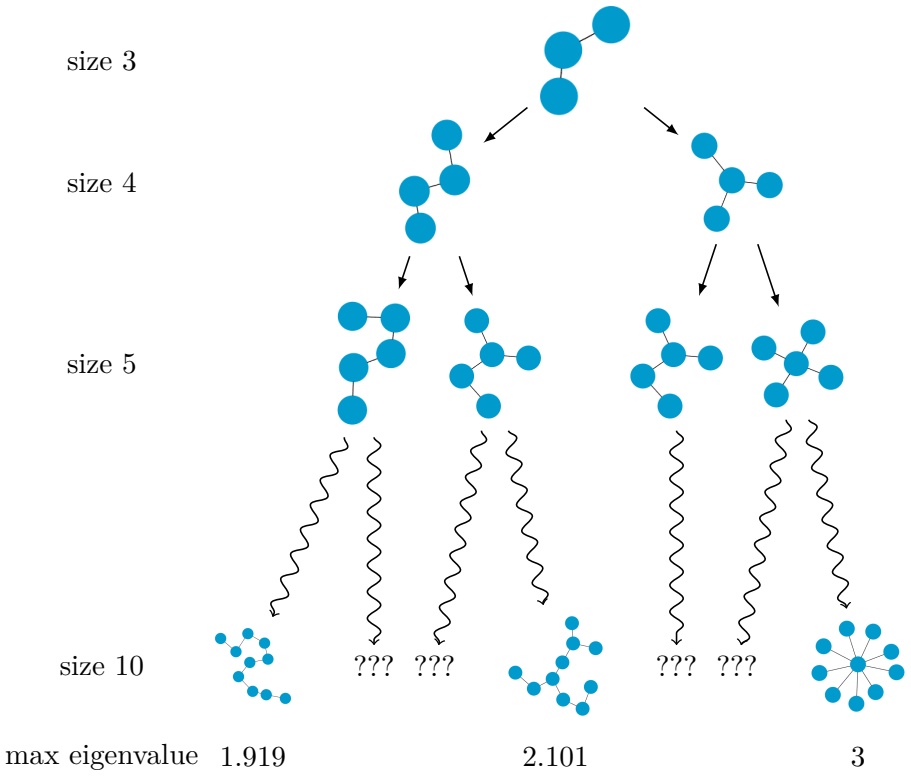


Figure 1.1: Search tree of the tree generation problem. The ondulated arrows represent playouts from size 5 graphs to terminal size 10 graphs. MCTS algorithm should direct themselves toward the right size of the search space tree during their search.

1.5 Plan

This thesis is disjointed and regroups multiple applications of Monte Carlo Search and other search algorithms on separate problems. But there are a few themes common to some of these applications.

The first theme is the usual computer scientist combinatorial problems, involving graphs (and coalition structures). These problems have multiple advantages: they are broad, abstract, and easily implementable, they are featured in multiple papers, they gather communities of researchers specialized in algorithmics and combinatorial optimization These advantages ensure a strong SOTA and a variety of approaches to compare to. They are a good ground to show the advantages or disadvantages of the methods we introduce.

The first problem we experimented on among these was the "automated spectral graph theory refutation". Graphs and algebra are two subjects I am deeply interested in. While MCTS (especially nested ones) outperformed a previous neural-based method and SOTA, greedy searches proved to be faster in most situations. Machine learning being outperformed by greedy search in graph algebra seems to be a result corroborated by Deepmind. (Mehrabian, A., Anand, A., Kim, H., Sonnerat, N., Balog, M., Comanici, G., ... & Wagner, A. Z. (2023). Finding increasingly large extremal graphs with AlphaZero and tabu search. arXiv preprint arXiv:2311.03583.)

The second graph-related problem was the budget-limited optimization of spatial transportation networks, where we optimized efficiency and robustness. Also outperforming the previous SOTA (UCT), efficiency was best optimized by a new algorithm of ours called Lazy Nested Monte Carlo Search, while the robustness was best optimized by a beam search (deterministic).

The final "usual computer scientist problem" we experimented on was the Coalition Structure Generation problem. In this problem agents must be placed in coalition(s) in order to maximize the sum of all the coalition values. On all variants of the problem, the previous SOTA (UCT with greedy playouts) was outperformed by either LNMCS (Lazy NMCS, a new variant of NMCS) with greedy playouts or with a new representation of the search space.

Another theme linking some of the works in this thesis is cheminformatics. While I do not have a formation in chemistry past high school, I always was passioned about molecules, atoms, and the structure of matter (it was a tough choosing between biology, physics, mathematics, and computer science). I am also interested in the idea of the method I am working on potentially helping people.

The first problem related to cheminformatics we experimented on is a toy model called the "Hydrophobic-Polar model". It is a toy model that could have been labeled as a "usual computer scientist problem' since it is abstract

and puzzle-like. It was invented to mimic the folding of real life proteins, and even if the application to chemistry may be limited, it's granular and difficult nature, and the old (and many) algorithms revolving around it helped us imagine the LNMCS, variant of the NMCS, outperforming all known MCTS algorithms on all problems we tried it on so far. It is also quite fun to play with.

The second problem is the much larger retrosynthesis problem. Retrosynthesis is a well-known problem that has been featured in hundreds of articles. It involves two parts: a neural network (one step retrosynthesis) and a search algorithm. By swapping the UCT algorithm with a NMCS in AstraZeneca's AiZynthFinder we are able to improve its performances noticeably.

The last problem is the perhaps even larger topic of de novo molecule design. We partnered with chemists and used ngrams extracted from an established molecule database, paired with an atom by atom SMILES based search space representation. This resulted in a software able to generate thousands of drug-like, valid, novel, and generally unique molecules per second. The molecules are also fairly (32%) synthesizable. To the best of our knowledge, no other software has these capacities.

The last theme is one of puzzle design. Puzzles are another of my topics of interest, I love playing them and designing them. I especially like nonograms, the last project in this thesis is an automated multi-purpose nonogram generator with difficulty estimation.

If you were to read only four papers of this thesis, we recommend "Solving the Hydrophobic-Polar Model with Nested Monte Carlo Search" for the introduction of the Lazy Nested Monte Carlo Search algorithm, "Lazy Nested Monte Carlo Search for Coalition Structure Generation" for a showcase of said algorithm, "Comparing search algorithms on the retrosynthesis problem" for improvements on a state of the art retrosynthesis tool, and last "DrugSynthMC: an atom based generation of drug-like molecules with Monte Carlo Search" for a search model that only produces valid molecules, and that is able to outperform the state of the art in many aspects with the use of ngrams instead of neural networks.

Here is a list of published papers:

1. Refutation of Spectral Graph Theory Conjectures with Monte Carlo Search
2. Lazy Nested Monte Carlo Search for Coalition Structure Generation
3. Solving the Hydrophobic-Polar model with Nested Monte Carlo Search
4. Comparing search algorithms on the retrosynthesis problem
5. DrugSynthMC: an atom based generation of drug-like molecules with Monte Carlo Search

6. Generating Difficult and Fun Nonograms

Chapter 2

Graphs and coalitions

2.1 Spectral graph theory conjectures

The attempt at refuting conjectures using Monte Carlo Tree Search algorithms was chronologically the first project in this thesis, and the first one to be published. This is why fewer algorithms were used, they were not implemented yet. It was initiated by a 2021 article from researcher Adam Zsolt Wagner using a deep reinforcement learning method called deep cross-entropy, which has since been cited more than 50 times as of 2024 and inspired a good number of papers. Our approach aimed at comparing Monte Carlo and deterministic search algorithms to state of the art deep learning over 3 conjectures, by constructing graphs edge by edge. The search algorithms outperformed and succeeded where deep learning failed.

Spectral graph theory is very interesting for multiple reasons. (1) Spectral graph conjectures are plenty thanks to automated spectral graph conjecture generators (2) the spectrum of a matrix is inexpensive to compute for small matrices (size < 50) (3) the spectrum of a matrix can be of interest for many applications such as internet/transportation network optimization.

The first article was published at the COCOON conference, we followed this research with another article trying a much larger set of algorithms over all conjectures from the software Graffiti that do not require computing NP hard invariants (parts of the conjecture like the chromatic number for example). All conjectures refuted in previous papers were also refuted by our algorithms, and another, previously open, conjecture was refuted. The greedy best-first search algorithm, a greedy search algorithm, was generally the best algorithm. This result is corroborated by recent findings in graph combinatorics from Deepmind, greedy algorithms and heuristics tend to fare better on graph theory problems.

Last, Liora Taieb applied our algorithms and code to 68 more computer-generated conjectures during her internship. These conjectures were previously refuted using a combination of using a special sample of subquartic

graphs (no vertex has a degree superior to 4), and the same deep cross-entropy method introduced by Wagner. All conjectures refuted by the deep cross entropy were also refuted by tree search methods, proving once again the superiority of our methods. In addition, 3 open conjectures, which were not refuted by hand, by exhaustive search, by subquartic graphs, and by deep reinforcement learning, were refuted.

It is interesting to note, that unlike with the Graffiti conjectures in the second article, the conjectures from Liora's third article are best refuted with NMCS and NRPA when a policy needs to be learned.

Refutation of Spectral Graph Theory Conjectures with Monte Carlo Search

Milo Roucairol and Tristan Cazenave

LAMSADE, Université Paris Dauphine - PSL, CNRS

Abstract. We demonstrate how Monte Carlo Search (MCS) algorithms, namely Nested Monte Carlo Search (NMCS) and Nested Rollout Policy Adaptation (NRPA), can be used to build graphs and find counter-examples to spectral graph theory conjectures in minutes.

Keywords: Monte Carlo Search · Graph Theory · Conjecture · Refutation.

1 Introduction

Monte Carlo search algorithm have proven to be powerful as game playing agents, with recent successes like AlphaGo[19]. These algorithms have the advantage of only needing an evaluation function for the final state of the space they explore.

Graph conjectures are propositions on graph classes (any graph, trees, K-free...) that are suspected to be true and are awaiting a proof or a refutation. They lend themselves well to computer assisted proofs, as finding a counter example can be tedious to do manually. Spectral graph conjectures are appropriate to automated refutation because the property can often directly be turned into the evaluation function that can take many different values. Thanks to software like Auto-GraphiX [18] and Graffiti [10], there are plenty of such conjectures.

Adam Zsolt Wagner showed that one could find explicit counter-examples using deep reinforcement learning of a policy with the deep cross entropy method [26].

In this paper, the MCS algorithms will play the game of refuting conjectures by building counter-examples.

First we will present the refutation of graph theory conjectures, then the different algorithms we use to explore the problem space, after that the procedure we use to build graphs and the game rules, finally we expose our results on four different conjectures.

2 Refutation of Graph Theory Conjectures

2.1 Graph Theory Conjectures

Conjectures in graph theory can be difficult to refute manually, unless one has an intuition of a counter-example, building a large number of graphs and computing invariant values or NP-hard problems on them often results in a waste of

time. Hopefully, computers can help us with these score computations, the goal of automated conjecture refutation is to automatize the exploration as well.

Multiple types of graph theory conjectures exist: existence, topological, flow based, connectivity, cycle, minors, spectral... The conjectures we are examining here are the spectral ones, they have the advantage of only requiring matrices calculations. The spectrum of a matrix is the set of its eigenvalues, spectral graph theory conjectures include conjectures on spectrum related invariants on different types of graph related matrices (adjacency, distance, laplacian...).

In this article we aim at evaluating the interest of Monte Carlo Search methods in order to refute some spectral graph theory conjectures. We are interested in conjectures that Adam Zsolt Wagner refuted in his article [26] as well as in other conjectures.

2.2 Algorithms Used to Refute Graph Conjectures

Wagner used deep neural networks with cross entropy. The network is used to learn a policy from a state, it is trained by constructing a batch of graphs according to the policy, evaluating and selecting the best graphs from the batch to adjust the neural network with their state-moves couples according to their scores (obtained during the evaluation), then it's repeated starting by the generation of a new batch until the conjecture is refuted.

Some of our conjectures come from Aouchiche and Hansen survey [1]. In order to explain how Graffiti selects conjectures they state:
"Graffiti generates many conjectures of a simple form (e.g. inequalities between two invariants or between an invariant and the sum of two others) then tests them on a database of graphs and discards those which are falsified. Should this test be passed, it is checked if the formulas are implied by known ones (in which case they are also discarded) and that they provide new information for at least one graph in the database, i.e., that they are stronger than the conjunction of all other formulas for that graph. If not, they are temporarily set aside. If yes, they are proposed to all graph theorists, in the electronic file "Written on the Wall", which reports on the status of almost 1000 conjectures. Many well-known graph theorists worked on these conjectures and this led to several dozen papers. Some of Graffiti's conjectures are about various topics in spectral graph theory, namely, the eigenvalues, as well as their multiplicity, of the adjacency, Laplacian and distance matrices of graphs."

3 Algorithms

In this section we present Monte Carlo Search and the different search algorithms we use: NMCS, NRPA and Greedy-BFS.

3.1 Monte Carlo Search

Monte Carlo Tree Search (MCTS) has been successfully applied to many games and problems [5].

Nested Monte Carlo Search (NMCS) [6] is an algorithm that works well for puzzles and optimization problems. It biases its playouts using lower level playouts. At level zero NMCS adopts a uniform random playout policy. Online learning of playout strategies combined with NMCS has given good results on optimization problems [24]. Other applications of NMCS include Single Player General Game Playing [20], Cooperative Pathfinding [3], Software testing [22], heuristic Model-Checking [23], the Pancake problem [4], Games [7] and the RNA inverse folding problem [21].

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) [25]. NRPA has found new world records in Morpion Solitaire and crosswords puzzles. NRPA has been applied to multiple problems: the Traveling Salesman with Time Windows (TSPTW) problem [8,12], 3D Packing with Object Orientation [14], the physical traveling salesman problem [15], the Multiple Sequence Alignment problem [16] or Logistics [13]. 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 [17].

3.2 Nested Monte Carlo Search

NMCS [6] is a Monte Carlo Search algorithm that 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, but it is not the case with the NMCS used here.

Algorithm 1 gives the NMCS algorithm. The different notations used are:

$M_{current\text{-}state}$ denotes the legal moves available from the state *current-state*.

$randomChoice(L)$ is a function that returns an element selected uniformly from L.

3.3 Nested Rollout Policy Adaptation

Introduced by Christopher D. Rosin [25], NRPA is akin to mixing NMCS [6] and Q-learning [27], performances are generally better than NMCS but the exploration/exploitation can lock itself in a local minimum. The algorithm is also dependent on how moves are defined.

The algorithm uses a policy which consist in attributing a value to a move. The difference with Wagner's policy [26] is here the value of the move is not necessarily learned given the state of the graph, but given the definition of the move, the policy can thus be defined given the few previous moves or the number of moves preceding the current one. It is also a key difference between NRPA policy learning and Q-learning, Wagner's policy is closer to deep Q-learning.

Like the NMCS, the NRPA calls other lower level NRPA which start with the higher level current policy, these lower level NRPA return their best sequence which is used to update the higher level policy. At the lowest level, the NRPA launches a playout and randomly select moves according to the policy.

$softmaxChoice(L, policy)$ is a function that returns an element selected according to a softmax distribution on each element value mapped on the policy. Chances of selecting e from L are $\frac{exp(policy[e])}{sum([exp(policy[i]) for\ i\ in\ L])}$.

In all our experiments, NRPA's *Alpha*, the learning rate, is set to 1.

Algorithm 2 gives the NRPA algorithm.

3.4 Greedy Best First Search

Another algorithm we use is Greedy-BFS [11]. It consists in opening the best (or randomly one of the best) previously evaluated node, evaluating all its children and inserting these children in a list, sorted by their evaluation scores.

The evaluation can be done with a heuristic (in which case it is not Monte Carlo) or a Monte Carlo algorithm like a playout or a low level NMCS.

In this paper, the Greedy-BFS was used as a general search technique and not a Monte Carlo search algorithm. We use the score function (or the modified evaluation function in conjecture 2) to choose which leaves to expand.

4 Graph Generation

For each problem we want to solve, we define an interface. This interface provides, to the search algorithm, the classes for the *state* and for the *move* and their associated functions: *score*, *terminal*, M_{state} and *play*.

The goal is to find an instance of a class of graphs that does not respect a property speculated on that class of graphs, in order to do so we use reinforcement learning and tree search algorithms to explore the possible graphs from that class and hopefully converge to a counter-example.

Moves to generate graphs are the same in all the approaches presented here, the graph is built edge by edge. Moves are represented as couple of integers corresponding to the two vertices the edge will link. If one of the two integer corresponding vertex is not in the graph yet, it is added alongside the edge as a leaf (-1 as the second member of the couple means adding a leaf anyway).

What differs from one conjecture (interface) to another are the methods :
- *score* which derives from the conjecture we want to refute.
- *terminal* is necessary to MCS methods to know when stop a playout, it can be the size of the graph (what we used here) but also the number of edges or any property on the graph. Trying different constraints for that method is part of the experiments.
- M_{state} gives the legal moves from a state according to the model. For example a model of maximum degree 3 will exclude any moves that result in the edge addition on a vertex of degree 3.

5 Experiments on Conjectures

The experiments were made with Rust 1.59, on a Intel Core i5-6600K 3.50GHz using a single core (but parallel processing is very accessible).

Every solution was verified using WolframAlpha symbolic computing of eigenvalues to ensure they were not due to floating point errors in Rust nalgebra 0.30.1 library.

We express every score function in order to maximise them.

5.1 Conjecture 1. AutoGraphiX

Conjecture 7 from [1], also present as conjecture 2.1 from [26], is used as a proof of concept in the aforementioned paper. We decided to refute all conjectures from [26] to show that MCS can be equally as good as the deep cross entropy.

Definition 1. *Let G be a connected graph, the matching number is the size of the matching that contains the largest number of edges of G.*

Definition 2. *Let G be a connected graph on n vertices, the distance spectrum $\lambda_1, ..., \lambda_n$ is the spectrum of the distance matrix, ranked in descending order (λ_1 being the largest, the index).*

This conjecture is the following :

Conjecture 1. *Let G be a connected graph on $n \geq 3$ vertices with index λ_1 and matching number μ. Then $\lambda_1 + \mu \geq \sqrt{n-1} + 1$.*

$\lambda_1 + \mu \geq \sqrt{n-1} + 1 \Leftrightarrow 0 \geq \sqrt{n-1} + 1 - \lambda_1 - \mu$

Which gives

Score function : $\sqrt{n-1} + 1 - \lambda_1 - \mu$

With a NRPA of level 3, and generating trees of size 19, we found a graph with a positive score of 0.016: $\lambda_1 + \mu \approx 5.227$ and $\sqrt{n-1} + 1 \approx 5.243$
It is the same counter-example (figure 1) as Adam Zsolt Wagner [26]. His results

helped us as we knew what to search for and that we could exploit NRPA policy learning to recreate the pattern found in his counter example. However it is interesting to try generating trees first anyway when refuting a spectral graph conjecture as the space to explore is way smaller than for any graph, and trees are extreme instances of the class, so these tend to be counter-examples.

The NRPA algorithm can lock itself in a policy leading to a local minimum while searching a counter-example to that conjecture. It is important to use restarts in this case: if a result is not found in less than a second the algorithm is launched again a few number of times (usually 10 to 20 times) until the refutation is found. While it takes a few hours to find the counter example with Wagner's deep cross entropy [26], our method can find it in seconds even with multiple relaunches. Using restarts this way can be parallelized easily.

We also were able to find a counterexample of size 18 (figure 2) with the same process, with a positive score of 0.012: $\lambda_1 + \mu \approx 5.101$ and $\sqrt{n-1} + 1 \approx 5.123$

5.2 Conjecture 2. Aouchiche-Hansen

Conjecture 2.15 from [2], also present as conjecture 2.3 from [26].

Definition 3. *Let G be a connected graph, the diameter is the maximum length of the shortest paths between all vertices of G.*

Definition 4. *Let G be a connected graph, the proximity is, over all the vertices of G, the minimum average distance to all the other vertices of G.*

Conjecture 2. *Let G be a connected graph on $n \geq 4$ vertices with diameter D, proximity π and distance spectrum $\lambda_1 \geq ... \geq \lambda_n$. Then $\pi + \lambda_{\lfloor \frac{2D}{3} \rfloor} \geq 0$.*

Results from Wagner paper helped us to know what kind of graph generate: a tree with more than 203 vertices. The intuition of that counter-example can also be seen in [2].

The NRPA algorithm, with the standard score function, finds trees of size 30 with similar scores (-0.4) as the deep cross entropy method [26] in 3s, when it takes days of training to the deep cross entropy to achieve these scores.

However none of the Monte Carlo method found a counter example with these information only, just like Wagner's deep cross entropy. We determined that the problem stemmed from the score function which lumps scores from different graphs close together, a graph can see its diameter increase but not see repercussions on its score due to the flooring (increasing the diameter of the graph is indeed the way to find a counter example). To solve this problem, we designed an evaluation function slightly different from the score function:

Score function: $-\pi - \lambda_{\lfloor \frac{2D}{3} \rfloor}$

Evaluation function: $-\pi - \lambda'_{2D}$

We define λ' by linearly interpolating λ to be 3 times as long.

With this new evaluation function, increasing the diameter of the graph always leads to a very different score. Instead of playouts, we then used that evaluation function in the greedy-BFS algorithm which found a counter example (figure 3) in about 5 minutes.

The graph shown in figure 3 had a score of approximately 0.000285, meaning $\pi + \lambda_{\lfloor \frac{2D}{3} \rfloor} < 0$, it's indeed a counter-example (it was verified to not be a floating point error with wolfram symbolic eigenvalues computing).

5.3 Conjecture 3. Collins

Conjecture 10 from [9], also present as conjecture 2.4 from [26].

Given a tree T on n vertices, its adjacency matrix $A(T)$ and its distance matrix $D(T)$.

$CPD(T)$ is the characteristic polynomial of $D(T)$:

$$CPD(T) = det(D(T) - Ix) = \sum_{k=0}^{n} \delta_k x^k$$

Let the coefficients $d_k = \frac{2^k}{2^{n-2}}|\delta_k|$ for k in 0, ..., n-2 be the normalized coefficient of the characteristic polynomial.

Let $p_D(T)$ be the peak of the normalized coefficient of the characteristic polynomial.

$CPA(T)$ is the characteristic polynomial of $A(T)$:

$$CPA(T) = det(A(T) - Ix) = \sum_{k=0}^{n} a_k x^k$$

Let $p_A(T)$ be the peak of the non-zero coefficients (a_k) of $CPA(T)$.

Conjecture 3. *Given a tree T, $CPA(T)$ form an unimodal sequence and its peak $p_A(T)$ is at the same place as $p_D(T)$.*

Like Adam Zsolt Wagner [26], we only refuted the second part of the conjecture with a level 2 NMCS, using the same score function.

Score function : $|\frac{p_A(T)}{\#\lambda_{\neq 0}} - (1 - \frac{p_D(T)}{n-2})|$

The counter-example in figure 4 was found in less than a second, it features a $p_A(T)$ at 19/32 and a $p_D(T)$ at 16/30 which gives no doubt that the peaks are at different positions.

5.4 Conjecture 4. Graffiti 137

That conjecture was made automatically by a program named Graffiti [10], it is present in the survey [1] as an already refuted conjecture.

Let $Hc = \sum_{uv \in E} \frac{1}{d(u)+d(v)}$ the harmonic of graph G with vertices E, $d(u)$ is the degree of the vertex u.

Conjecture 4. *For any graph G, the second biggest adjacency matrix eigenvalue λ_2 is inferior to the harmonic of the graph : $\lambda_2 < Hc(G)$.*

Score function : $\lambda_2 - Hc(G)$

A counter-example of size 7 (figure 5) is quickly found by NMCS of level 2, NRPA of level 2 or even Greedy-BFS with $\lambda_2 \approx 1.786$ and $Hc(G) = 5/3 \approx 1.576$.

5.5 Comparison of Search Algorithms

Table 1 gives the times required to refute each conjecture for each algorithm. The machine used has a i5-6600K 3.5 GHz CPU. We see that NMCS can refute conjectures 3 and 4 almost instantly but does not find refutations to conjectures 1 and 2. NRPA also refutes conjectures 3 and 5 almost instantly and refutes conjecture 1 in 0.1 seconds when Wagner solves it in a few hours. Greedy-BFS refutes conjectures 2 and 4. It is the only algorithm (among those we tried) able to refute conjecture 2, using a modified score function to guide a non MC search. It solves it in 5 minutes when Wagner algorithm solves it in a few days.

Conjecture 2 could not be refuted with the natural score function derived from the conjecture but could be solved with Greedy-BFS using a more informative score function. NRPA of level 2 with the natural score function could attain a score of -0.4 in 3s instead of a few days in Wagner's work [26].

6 Conclusion

Monte Carlo search methods proved to be powerful ways of refuting conjecture from spectral graph theory much faster than Wagner's deep cross entropy method [26]. Trying to build trees even when the conjecture is applied on any graph can also be helpful as it reduces the amount of possible builds greatly, it is inexpensive and should be tried first.

Table 1. Summary of our results

Method	Conj 1	Conj 2*	Conj 3	Conj 4
NMCS	-	-	1s (lv 2)	0s (lv 2)
NRPA	1s (lv 3)	-	1s (lv 2)	0s (lv 2)
Greedy-BFS	-	291s	-	0s

(-) denotes a failure, an inability to refute the conjecture.
* Refuted using the evaluation function, different from the score function, that evaluates non terminal states.

However, these methods present limits. Computing score functions that require eigenvalues on big trees (over size 500) can be very costly. They are also dependent on the shape of the score function: a noisy score function with many local minimum can be challenging, as well as a score function with more discrete results can lead to an absence of differentiation in the paths to explore (see conjecture 2). Conjectures requiring to compute a NP hard problem can also severely increase the computing time even for small graphs (30 vertices).

In the future we aim to refute more conjectures and to improve the Greedy-BFS with MCS method, potentially with pruning strategies too.

References

1. Aouchiche, M., Hansen, P.: A survey of automated conjectures in spectral graph theory. Linear Algebra and its Applications **432**(9), 2293–2322 (Apr 2010). https://doi.org/10.1016/j.laa.2009.06.015, `https://www.sciencedirect.com/science/article/pii/S0024379509003061`
2. Aouchiche, M., Hansen, P.: Proximity, remoteness and distance eigenvalues of a graph. Discrete Applied Mathematics **213**, 17–25 (Nov 2016). https://doi.org/10.1016/j.dam.2016.04.031, `https://www.sciencedirect.com/science/article/pii/S0166218X16302037`
3. Bouzy, B.: Monte-carlo fork search for cooperative path-finding. In: Computer Games Workshop at IJCAI. pp. 1–15 (2013)
4. Bouzy, B.: Burnt pancake problem: New lower bounds on the diameter and new experimental optimality ratios. In: SOCS. pp. 119–120 (2016)
5. Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of Monte Carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in Games **4**(1), 1–43 (Mar 2012). https://doi.org/10.1109/TCIAIG.2012.2186810
6. Cazenave, T.: Nested Monte-Carlo Search. In: Boutilier, C. (ed.) IJCAI. pp. 456–461 (2009)
7. Cazenave, T., Saffidine, A., Schofield, M.J., Thielscher, M.: Nested monte carlo search for two-player games. In: AAAI. pp. 687–693 (2016)

8. Cazenave, T., Teytaud, F.: 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. pp. 42–54 (2012)
9. Collins, K.L.: On a conjecture of Graham and Lovász about distance matrices. Discrete Applied Mathematics **25**(1), 27–35 (Oct 1989). https://doi.org/10.1016/0166-218X(89)90044-9, https://www.sciencedirect.com/science/article/pii/0166218X89900449
10. Delavina, E.: Some History of the Development of Graffiti. In: DIMACS Series in Discrete Mathematics and Theoretical Computer Science 69: Graphs and Discovery. pp. 81–118
11. Doran, J.E., Michie, D.: Experiments with the graph traverser program. Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences **294**(1437), 235–259 (1966)
12. Edelkamp, S., Gath, M., Cazenave, T., Teytaud, F.: Algorithm and knowledge engineering for the tsptw problem. In: Computational Intelligence in Scheduling (SCIS), 2013 IEEE Symposium on. pp. 44–51. IEEE (2013)
13. Edelkamp, S., Gath, M., Greulich, C., Humann, M., Herzog, O., Lawo, M.: Monte-Carlo tree search for logistics. In: Commercial Transport, pp. 427–440. Springer International Publishing (2016)
14. Edelkamp, S., Gath, M., Rohde, M.: Monte-Carlo tree search for 3d packing with object orientation. In: KI 2014: Advances in Artificial Intelligence, pp. 285–296. Springer International Publishing (2014)
15. Edelkamp, S., Greulich, C.: Solving physical traveling salesman problems with policy adaptation. In: Computational Intelligence and Games (CIG), 2014 IEEE Conference on. pp. 1–8. IEEE (2014)
16. Edelkamp, S., Tang, Z.: Monte-Carlo tree search for the multiple sequence alignment problem. In: Proceedings of the Eighth Annual Symposium on Combinatorial Search, SOCS 2015. pp. 9–17. AAAI Press (2015)
17. Finnsson, H., Björnsson, Y.: Simulation-based approach to general game playing. In: Aaai. vol. 8, pp. 259–264 (2008)
18. Hansen, P., Caporossi, G.: AutoGraphiX: An Automated System for Finding Conjectures in Graph Theory. Electronic Notes in Discrete Mathematics **5**, 158–161 (Jul 2000). https://doi.org/10.1016/S1571-0653(05)80151-9, https://www.sciencedirect.com/science/article/pii/S1571065305801519
19. Holcomb, S.D., Porter, W.K., Ault, S.V., Mao, G., Wang, J.: Overview on DeepMind and Its AlphaGo Zero AI. In: Proceedings of the 2018 International Conference on Big Data and Education. pp. 67–71. ACM, Honolulu HI USA (Mar 2018). https://doi.org/10.1145/3206157.3206174, https://dl.acm.org/doi/10.1145/3206157.3206174
20. Méhat, J., Cazenave, T.: Combining UCT and Nested Monte Carlo Search for single-player general game playing. IEEE Transactions on Computational Intelligence and AI in Games **2**(4), 271–277 (2010)
21. Portela, F.: An unexpectedly effective Monte Carlo technique for the RNA inverse folding problem. BioRxiv p. 345587 (2018)
22. Poulding, S.M., Feldt, R.: Generating structured test data with specific properties using nested Monte-Carlo search. In: GECCO. pp. 1279–1286 (2014)
23. Poulding, S.M., Feldt, R.: Heuristic model checking using a Monte-Carlo tree search algorithm. In: GECCO. pp. 1359–1366 (2015)
24. Rimmel, A., Teytaud, F., Cazenave, T.: Optimization of the Nested Monte-Carlo algorithm on the traveling salesman problem with time windows. In: EvoApplications. LNCS, vol. 6625, pp. 501–510. Springer (2011)

25. Rosin, C.D.: Nested rollout policy adaptation for Monte Carlo Tree Search. In: IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence. pp. 649–654 (2011)
26. Wagner, A.Z.: Constructions in combinatorics via neural networks. arXiv:2104.14516 [cs, math] (Apr 2021), http://arxiv.org/abs/2104.14516, arXiv: 2104.14516
27. Watkins, C.J.C.H., Dayan, P.: Q-learning. Machine Learning **8**(3), 279–292 (May 1992). https://doi.org/10.1007/BF00992698, https://doi.org/10.1007/BF00992698

A Appendix - Algorithms

Algorithm 1 The NMCS algorithm.

```
NMCS (current-state, level)
if level = 0 then
    ply ← 0
    seq ← {}
    while current-state is not terminal do
        move ← randomChoice(M_current-state)
        current-state ← play(current-state, move)
        seq[ply] ← move
        ply+ = 1
    end while
     return score (current-state, seq)
else
    best-score ← −∞
    while current-state is not terminal do
        for each move in M_current-state do
            next-state ← play(current-state, move)
            (score, seq) ←NMCS (next-state, level − 1)
            if score ≥ best-score then
                next-best-state ← next-state
                best-score ← score
                best-sequence ← seq
            end if
        end for
        current-state ← next-best-state
    end while
     return (best-score, best-sequence)
end if
```

Algorithm 2 The NRPA algorithm.

NRPA (*policy*, *level*)
if $level = 0$ **then**
  $current\text{-}state \leftarrow root()$
  $ply \leftarrow 0$
  $seq \leftarrow \{\}$
  while *current-state* is not terminal **do**
    $move \leftarrow softmaxChoice(M_{current\text{-}state}, policy)$
    $current\text{-}state \leftarrow play(current\text{-}state, move)$
    $seq[ply] \leftarrow move$
    $ply+ = 1$
  end while
  return (score (*current-state*), *seq*)
else
  $best\text{-}score \leftarrow -\infty$
  for *N iterations* **do**
    $(result, new) \leftarrow$ NRPA($policy$, $level - 1$)
    if $result \geq best\text{-}score$ **then**
      $best\text{-}score \leftarrow result$
      $seq \leftarrow new$
    end if
    $pol \leftarrow adapt(pol, seq)$
  end for
  return $(best\text{-}score, seq)$
end if

Adapt (*policy*, *seq*)
$node \leftarrow root()$
$pol' \leftarrow pol$
for $ply = 0$ TO $length(seq) - 1$ **do**
  $pol'[(node, seq[ply])] \mathrel{+}= Alpha$
  $z \leftarrow Sum([exp(pol[(node, m)])$ for m in $M_{node}])$
  for each *move* in M_{node} **do**
    $pol'[(node, move)]$ -= $\frac{Alpha \cdot exp(pol[(node,move)])}{z}$
  end for
  $node \leftarrow play(node, seq[ply])$
end for
return pol'

B Appendix - Graphs

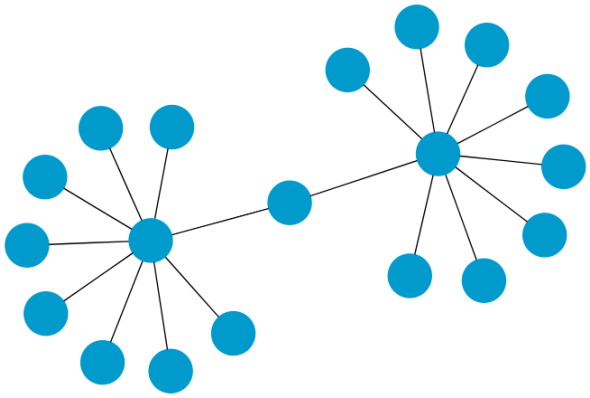


Fig. 1. Counter example of conjecture 1 of size 19

Edges: 0-1, 0-2, 0-3, 0-4, 0-5, 0-6, 0-9, 0-11, 0-18, 4-7, 7-8, 7-10, 7-12, 7-13, 7-14, 7-15, 7-16, 7-17

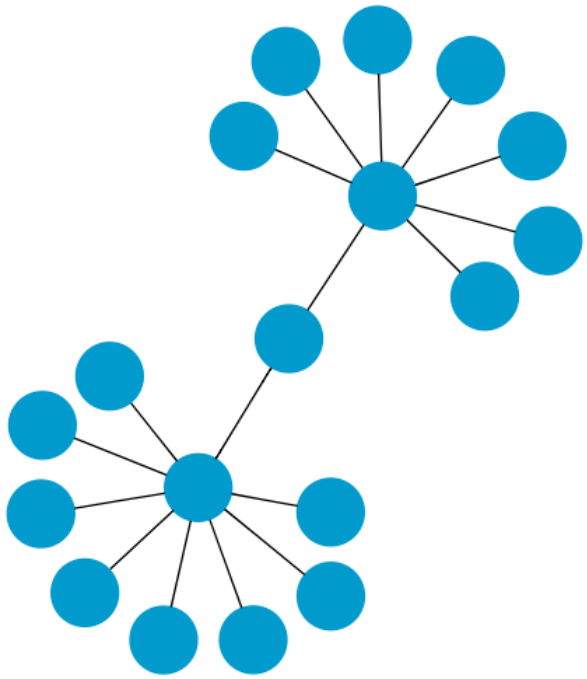


Fig. 2. Counter example of conjecture 1 of size 18

Edges: 0-1, 0-2, 0-3, 0-4, 0-5, 0-6, 0-9, 0-11, 4-7, 7-8, 7-10, 7-12, 7-13, 7-14, 7-15, 7-16, 7-17

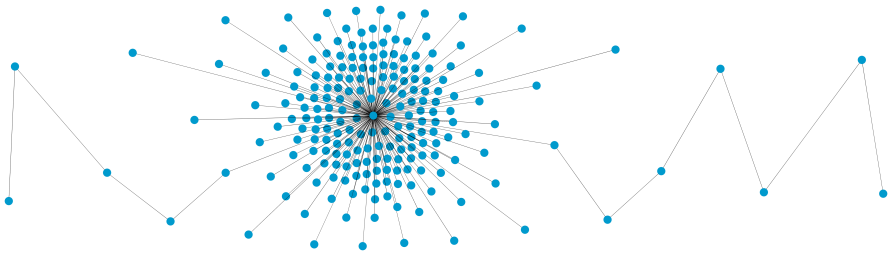


Fig. 3. Counter example of conjecture 2

Edges: 0-1, 0-8, 0-13, 0-14, 0-15, 0-16, 0-17, 0-18, 0-19, 0-20, 0-21, 0-22, 0-23, 0-24, 0-25, 0-26, 0-27, 0-28, 0-29, 0-30, 0-31, 0-32, 0-33, 0-34, 0-35, 0-36, 0-37, 0-38, 0-39, 0-40, 0-41, 0-42, 0-43, 0-44, 0-45, 0-46, 0-47, 0-48, 0-49, 0-50, 0-51, 0-52, 0-53, 0-54, 0-55, 0-56, 0-57, 0-58, 0-59, 0-60, 0-61, 0-62, 0-63, 0-64, 0-65,

0-66, 0-67, 0-68, 0-69, 0-70, 0-71, 0-72, 0-73, 0-74, 0-75, 0-76, 0-77, 0-78, 0-79, 0-80, 0-81, 0-82, 0-83, 0-84, 0-85, 0-86, 0-87, 0-88, 0-89, 0-90, 0-91, 0-92, 0-93, 0-94, 0-95, 0-96, 0-97, 0-98, 0-99, 0-100, 0-101, 0-102, 0-103, 0-104, 0-105, 0-106, 0-107, 0-108, 0-109, 0-110, 0-111, 0-112, 0-113, 0-114, 0-115, 0-116, 0-117, 0-118, 0-119, 0-120, 0-121, 0-122, 0-123, 0-124, 0-125, 0-126, 0-127, 0-128, 0-129, 0-130, 0-131, 0-132, 0-133, 0-134, 0-135, 0-136, 0-137, 0-138, 0-139, 0-140, 0-141, 0-142, 0-143, 0-144, 0-145, 0-146, 0-147, 0-148, 0-149, 0-150, 0-151, 0-152, 0-153, 0-154, 0-155, 0-156, 0-157, 0-158, 0-159, 0-160, 0-161, 0-162, 0-163, 0-164, 0-165, 0-166, 0-167, 0-168, 0-169, 0-170, 0-171, 0-172, 0-173, 0-174, 0-175, 0-176, 0-177, 0-178, 0-179, 0-180, 0-181, 0-182, 0-183, 0-184, 0-185, 0-186, 0-187, 0-188, 0-189, 0-190, 0-191, 0-192, 0-193, 0-194, 0-195, 0-196, 0-197, 0-198, 0-199, 0-200, 0-201, 0-202, 1-2, 2-3, 3-4, 4-5, 5-6, 6-7, 8-9, 9-10, 10-11, 11-12

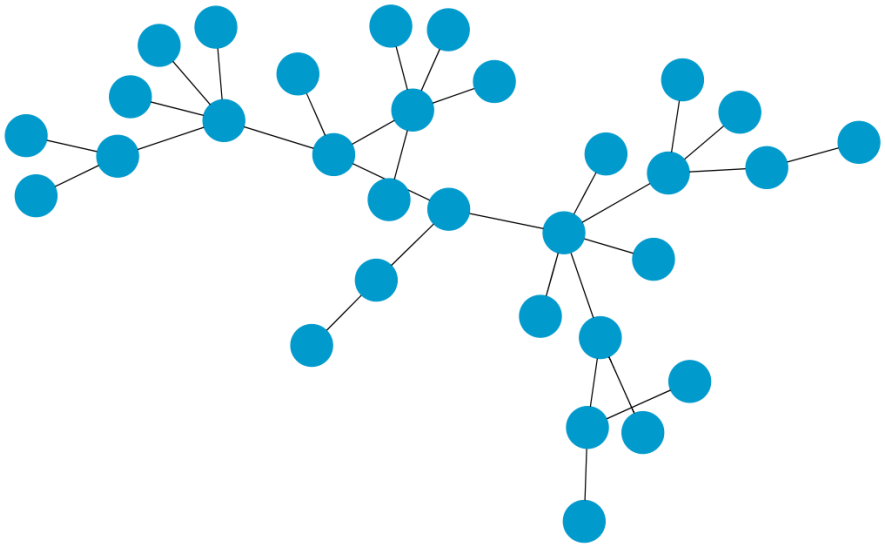


Fig. 4. Counter example of conjecture 3

Edges: 0-1, 0-2, 0-13, 0-27, 1-5, 1-7, 1-9, 1-22, 2-3, 3-4, 3-8, 3-12, 4-10, 4-25, 5-6, 5-16, 5-21, 6-14, 6-28, 9-11, 10-20, 11-24, 13-19, 14-15, 15-17, 15-18, 15-23, 19-26, 19-30, 28-29

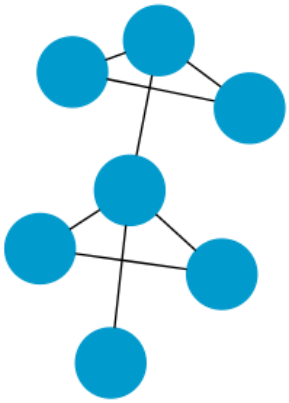


Fig. 5. Counter example of conjecture 4

Edges: 0-1, 0-2, 1-2, 0-3, 3-4, 4-5, 3-5, 3-7

Refutation of Spectral Graph Theory Conjectures with Search Algorithms

Milo Roucairol[a] and Tristan Cazenave[a]

[a]LAMSADE, Université Paris-Dauphine, Paris, Pl. du Maréchal de Lattre de Tassigny, 75016 Paris, France

Abstract. We are interested in the automatic refutation of spectral graph theory conjectures. Most existing works address this problem either with the exhaustive generation of graphs with a limited size or with deep reinforcement learning. Exhaustive generation is limited by the size of the generated graphs and deep reinforcement learning takes hours or days to refute a conjecture. We propose to use search algorithms to address these shortcomings to find potentially large counter-examples to spectral graph theory conjectures in seconds. We apply a wide range of search algorithms to a selection of conjectures from Graffiti. Out of 13 already refuted conjectures from Graffiti, our algorithms are able to refute 12 in seconds. We also refute conjecture 197 from Graffiti which was open until now.

Keywords: Monte Carlo Search · Spectral · Graph Theory · Conjecture · Refutation.

1 Introduction

Monte Carlo search algorithms have proven to be powerful as game-playing agents, with recent successes like AlphaGo [14]. These algorithms have the advantage of only needing an evaluation function for the final state of the space they explore.

Graph conjectures are propositions on graph classes (any graph, trees, $K_n - free$...) that are suspected to be true and are awaiting proof or a refutation. They lend themselves well to computer-assisted proofs, as finding a counter-example can be tedious to do manually. Spectral graph conjectures are appropriate for automated refutation because the property can often directly be turned into the evaluation function that can take many different values. Thanks to software like Auto-GraphiX [9] and Graffiti [5], there are plenty of such conjectures.

Adam Zsolt Wagner showed that one could find explicit counter-examples using deep reinforcement learning of a policy with the deep cross entropy method [16]. Wagner was able to refute three conjectures from Graffiti. Wagner's method trained a neural network with reinforcement learning for each conjecture, learning to build graphs tailored to the refutation of each conjecture. The refutation of a conjecture with Wagner's method takes hours or days. Using search algorithms it was possible to refute conjectures 2.1 and 2.4 from the Wagner paper in one second and conjecture 2.3 in 291 seconds [12].

A usual way to find counter-examples to graph theory conjectures is to generate all the graphs smaller than a given size and to verify the conjecture for all these small graphs [1]. This exhaustive generation limits the size of the graphs that can be tested. On the contrary, our search algorithms can rapidly generate larger refutation graphs.

In this paper, the search algorithms will play the game of refuting conjectures by building counter-examples edge by edge. We build on our previous paper [12] by testing many more conjectures and by comparing more search algorithms.

First, we will present the refutation of graph theory conjectures, then the different algorithms we use to explore the problem space, after that the procedure we use to build graphs and the game rules, and finally we expose our new results on multiple different conjectures.

2 Refutation of Graph Theory Conjectures

2.1 Graph Theory Conjectures

Refuting Graph theory conjectures can be a hard task to do manually. Imagining a large number of graphs and computing invariant or NP-hard problem values can be tedious and often results in a waste of time. Computers are designed to help with these score computations. The goal of automated conjecture refutation is to automatize the exploration as well.

Graph theory conjecture can concern many different properties of graphs: existence, topology, flow, connectivity, cycle, minors, spectral... Here we are interested in the spectral graph theory conjectures, necessitating matrix calculations only. The spectrum of a matrix is the set of its eigenvalues. Spectrum-related invariants on different types of graph-related matrices (adjacency, distance, Laplacian...) are the focus of spectral graph theory conjectures.

We propose to compare search algorithms for the refutation of various spectral graph theory conjectures. In [12], the conjectures from [16] were refuted hundreds of times faster and the performances of the methods were comapred. This paper experiments with a more diverse set of search algorithms and we address all the remaining conjectures from Graffiti that do not involve solving a NP-hard problem. Graffiti is a well-known software that was used to generate a thousand spectral graph theory conjectures. It was the target of numerous works and discussions, sometimes involving some of the greatest graph theoricians such as Paul Erdos, Laszlo Lovasz, Noga Alon, Noam Nisan. Conjectures from Graffiti were the subject of entire articles, with graph theorists dedicating weeks or months to them. Our program aims at refuting some of them in a much more efficient manner.

We decided to focus on this subset of Graffiti conjectures to avoid dependence on other algorithms and to reduce the number of conjectures.

2.2 Algorithms Used to Refute Graph Conjectures

The program by Wagner trained a deep neural network with cross-entropy to output a policy. The network is used to learn a policy from a state. The policy is used the generate a batch of graphs, it then evaluates the best graphs from the batch and trains the neural network using the graph's scores. Batches are successively evaluated until the refutation of the conjecture.

Softwares used for automated conjecture generation like Graffiti or AutoGraphiX also have their own ways of automatically refuting conjectures to verify the easily refutable ones. Graffiti and its conjectures are detailed in Aouchiche and Hansen survey [1]. It generates many conjectures in the form of inequalities between invariants. It then tests these conjectures on a database of graphs and discards the falsified ones. Then it checks if the inequalities are not implied by already known theorems and conjectures. If a conjecture passes these tests, it is proposed to scientists. "Written on the wall" collects almost 1000 conjectures from Graffiti, along with the discussions of many scientists. Graffiti's conjectures connect various topics in spectral graph theory such as eigenvalues, adjacency, distance; Laplacian, and gravity matrices of graphs. These conjectures are the subjects of dozens of articles.

3 Materials and Methods

We decided to expand the experiments from [12] by using the same algorithms and adding two widely used Monte Carlo Search algorithms, another recent one, and another baseline simple algorithm on more conjectures produced by Graffiti [1]. The search model used by the algorithms to build the graphs is the same as in [12]: starting from a single node, new nodes and edges are added to the graph until it reaches a target size. These additions of nodes and edges are the edges of the search tree.

- NMCS [3] uses nested levels of search with random search at the base level. It is already used in [12].
- NRPA [11] learns a playouts policy with nested levels of best sequences. At the lowest level it makes playouts with the learned policy. It is already used in [12].
- GBFS is a simple greedy algorithm opening the best state from a list, evaluating the children of this state, and inserting these children in the list according to their evaluation. It is already used in [12].
- BEAM search, another baseline greedy search heuristic, keeps the *width* best states after each step of expanding and going down the search tree.
- Upper Confidence bounds applied to Trees (UCT) as described by Kocsis and Szepesvári [10], which is the most widely used MCTS algorithm, and close to PUCT used in outstanding works such as Deepmind's Alphazero [15].

- Rapid action value estimation (RAVE), a MCTS algorithm inspired by UCT and proposed by Siver and Gelly [8], then generalized as GRAVE by Cazenave [4] using a threshold on the number of playouts.
- Lazy Nested Monte Carlo Search (LNMCS), a variant of NMCS proposed by Roucairol and Cazenave [13] addresses a shortcoming of the NMCS. Before launching a costly lower level LNMCS on a child state, it is evaluated with playouts and pruned if the evaluation is not satisfying enough, this way it avoids sinking the time budget in fruitless subtrees and allows the LNMCS to use higher starting levels than the NMCS.

These conjectures (in table 1) were selected because they did not require NP-hard problems to be solved to compute the score, thus lending themselves well to Monte Carlo Search and other optimization methods. All of the fitting conjectures from Aouchiche and Hansen's survey [1] are included in our experiments. It is possible to tackle conjectures needing a NP-hard problem to be solved (like coloration) up to sizes 25-30, we might come back to such conjectures in future works.

4 Results

The experiments were made with Rust 1.59, on an Intel Core i5-6600K 3.50GHz using a single core (but parallel processing is very accessible).

Our positive results in table 1 were obtained by launching each algorithm on each conjecture with a time budget of 15 minutes. In the Graffiti column, R means the conjecture was already refuted, and O means the conjecture is still open.

- The NMCS used a level of 3.
- The LNMCS used a level of 4, 3 playouts per evaluation, and a ratio of 0.8.
- The UCT used a constant of 1.
- The GRAVE and RAVE used a reference of 5.
- The BEAM search used a width of 10.

5 Discussion

As shown in table 1, we were unable to refute conjectures that were open except one (either 197 or 322). This means either that our algorithms are not strong enough, or that these conjectures are true. However, with the exception of Graffiti 140, most of the already refuted conjectures we tried were refuted by at least one algorithm among GRAVE, RAVE, NMCS, LNMCS, NRPA, GBFS, and UCT. Graffiti 140 refutation seems to be lost, 140 is said to be refuted by another work by Favaron which we are unable to find.

Considering nonterminal states for evaluation helped immensely with the harder Graffiti 137, 139, 189, 289, and 301. Graffiti 289's repeating pattern may be an explanation as to why the NRPA succeeded where most other algorithms

failed. The times, when displayed in the hundreds of seconds, are approximations, only GBFS and BEAM are deterministic so the time is exact. In addition, NMCS, UCT, GRAVE, and RAVE were very unreliable at finding a counter-example for Graffiti 29, about once every 3 runs, when LNMCS was able to find a counter-example in less than 60 seconds in 9 out of 10 runs.

For Graffiti 189 and 195, we first searched in the "any graph" search space (instead of the type of graphs the conjecture is about), and then verified if the graph in question complied with the conjecture definition.

We think the relative failure of GBFS on Graffiti 30 is partly due to the score function leading the algorithm to open nodes to smaller graphs first, while the MCTS algorithms try larger graphs immediately using playouts. Beam search requires a width of 80 or more to find a counter example to this conjecture, so trying larger graphs is not sufficient on its own.

The beam search is very dependent on its width parameter, too high and it will not be able to explore deep in the search tree and find large counter examples (on Graffiti 137), and too low and it will not consider inferior intermediate states that would in fact lead to a counter example (on Graffiti 29 and 30). That makes it an inferior choice for spectral graph conjecture refutation compared to GBFS.

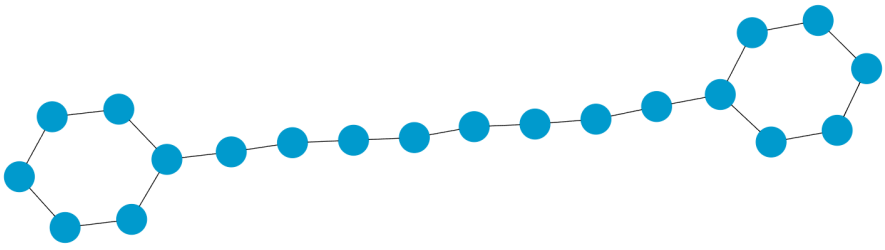


Fig. 1. A counter-example of Graffiti 289 of size 20 (second largest eigenvalue $\leq$ mean of the mean of all adjacent vertex degree for all nodes)
Edges: 0-1, 1-2, 2-3, 3-4, 4-5, 5-6, 6-7, 7-8, 8-9, 9-10, 10-5, 0-11, 11-12, 12-13, 13-14, 14-15, 15-16, 16-17, 17-18, 18-19, 19-14

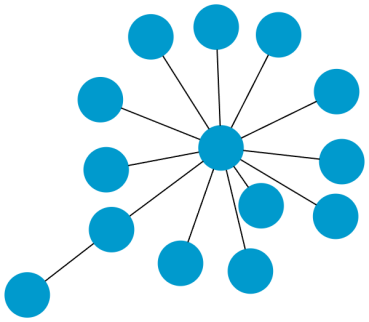


Fig. 2. A counter-example of Graffiti 301 of size 14 (scope of positive eigenvalues $\leq$ harmonic)
Edges: 0-1, 0-2, 1-3, 1-4, 1-5, 1-6, 1-7, 1-8, 1-9, 1-10, 1-11, 1-12, 1-13

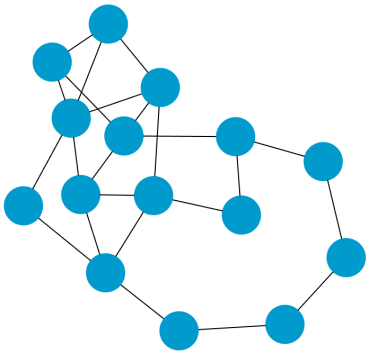


Fig. 3. A counter-example of Graffiti 30 of size 15 (# positive distance eigenvalues $\leq$ sum of temperatures)
Edges: 0-1, 0-3, 0-12, 1-2, 1-3, 2-3, 2-7, 2-12, 3-4, 3-5, 4-6, 5-6, 5-7, 5-12, 6-7, 6-8, 7-9, 8-10, 9-11, 10-14, 11-12, 11-13, 13-14

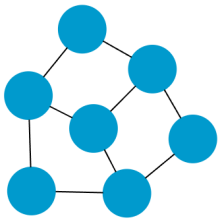


Fig. 4. A counter-example of Graffiti 29 of size 7 (randic index $\leq$ # negative eigenvalues)
Edges: 0-1, 0-2, 0-6, 1-3, 2-5, 3-4, 3-6, 4-5, 5-6

In[12] Graffiti 137 was one of the 4 main conjectures refuted. Unfortunately an erroneous definition of the Harmonic was used (see subsection 5.2). Graffiti 137 is actually more interesting than we initially thought, the smallest known counter-example featured 101 nodes and required a very specific structure, it was first refuted by Favaron et al. [6] using a mathematical proof and not directly a counter-example. Here we managed to find a counter-example of size 67 (see figure 5) which does not follow the structure described, but closely resembles it, by descending the search space (opening graphs of increasing size, much like conjecture 2) almost directly with GBFS.

As mathematical proofs are complex, if the goal was to refute this one conjecture, then our algorithm was most likely faster. But another interesting result is that the class of graph needed to refute Graffiti 137 by Favaron et al. was also used to refute two other conjectures (and could have been used to refute Graffiti 139, see figure 6), the search algorithms can provide insight to the mathematicians (they can provide insight even if the graph produced does not refute a conjecture as in Wagner's work [16] with conjecture 2.3). The other algorithms did not manage to find a counter-example in a reasonable time.

5.1 Graffiti 197 is refuted

In "Written on the wall", Graffiti 197 is defined as:

Graffiti 197. - 2-nd smallest eigenvalue $\leq$ range of eigenvalues of gravity matrix.

Using the usual definition of the range, and not Aouchiche and Hansen's (see subsection 5.2), our algorithms find a counter example with the cycle of length 17, see figure 7. Using Aouchiche and Hansen's definition of the range we could not find any counter example, but if this initial definition was to be used then Graffiti 322 would be (too) easily refuted.

The second smallest eigenvalue is $\lambda_{n-1} \approx -1.9659$, and the range of eigenvalues of the gravity matrix is approximately 1.7035.

Cycles of size 21 and 25 also refute Graffiti 197, we conjecture that any cycle of size $1 + 4 * n$ greater or equal to 17 refutes this conjecture.

5.2 Erratum

We provide a copy of "Written on the Wall" on this project's github, which is otherwise hard to find:

- Graffiti 290 (open) was solved instantly using the definition of gravity in Aouchiche and Hansen's survey [1] (which prompted us to verify the definition). But it was seemingly impossible using the definition from Brewster et al. [2].
  Since the definition of gravity of a graph is not widespread, here is the correct definition of the gravity matrix from "Written on the Wall" (page 52) and Brewster et al. [2]:

  Gravity matrix: The matrix (indexed by vertices of the graph) whose $(u, v)th$ entry is 0 if $u = v$ or if there is no path joining u to v; otherwise it is

  $$Gr(G) = (1/(n-1))(d(u) * d(v))/d(u, v) \tag{1}$$

  Where $d(u)$ is the degree of vertex u and $d(u, v)$ is the distance from vertex u to vertex v in the graph G of size n.
- In the same manner, with Aouchiche and Hansen's definition of the harmonic, Graffiti 140 was refuted by the graph with one edge connecting two vertices. The correct definition is featured in "Written on the Wall" (page 28) and in Favaron et al. [6]:

  Harmonic: The harmonic of a graph is defined as:

  $$Hc(G) = \sum_{uv \in E} \frac{2}{d(u) + d(v)} \tag{2}$$

  Where and E is the set of edges of a graph G.

- Using Aouchiche and Hansen survey's [1] definition of range, it appeared that our algorithms managed to refute Graffiti 322 (open) with a cycle graph of size 4. Here is the definition of Graffiti 322 from "written on the wall":

  Graffiti 322. If G is a triangle-free graph then the Inverse Even $\leq$ range of eigenvalues of Distance.

  Inverse Even is defined as the sum over the vertices of the inverse of the number of vertices to even distance from that vertex, ie:

$\sum_{v \in V} 1/Ev(v)$ where $Ev(v)$ is the number of vertices at even distance from vertex v.
Thus the cycle of size 4 has 3 distinct distance eigenvalues and an Inverse Even of 4. An error with the definitions seems more likely than this conjecture being left open after dozens of articles with such a simple counter-example. The range is defined as the number of distinct values in a vector in Aouchiche and Hansen's survey and (likely inherited from) Favaron et al.'s work, with no further sources. The range of a vector is usually the difference between the minimum and maximum values. Thus we think this definition is erroneous but we were not able to find the actual definition of the range used in "written on the wall". The results featured in table 1 for Graffiti 322 use the usual definition of range, either way, Graffiti 197 is refuted or Graffiti 322 is refuted.
- Graffiti 294 from Aouchiche and Hansen's survey is Graffiti 295 from "written on the wall", we go by this identifier in table 1.
- Graffiti 140 is said to be refuted by Favaron et al. in "residue of a graph" [7] according to "written on the wall", but there is no mention of "harmonic", "deviation" or Graffiti 140 in that paper.

The general lack of sourcing, the loss of multiple articles, and the misleading or erroneous definitions are detrimental to future work on Graffiti conjectures. For instance, we suspect that Graffiti conjectures 28 and 209 may not be refuted as sources are nonexistent (28) or seemingly lost (209). We decided to include Graffiti 140 in the table unlike 28 and 209 because we could find its refutation source, but we could not find its appearance in the source. Auto-GraphiX conjectures may be more favorable to continuing this work.

6 Conclusion

Search algorithms proved to be powerful ways of refuting conjecture from spectral graph theory, much faster than Wagner's deep cross entropy method [16]. We can identify three algorithms that seem more effective on the conjectures tested: GBFS which is overall faster but was not able to refute conjectures in [12] and struggled on Graffiti 30, NRPA which is very efficient when a repeating pattern refutes a conjecture, and LNMCS/NMCS which managed to be slightly faster than NRPA on Graffiti 29 and 301 while being faster than the other MCTS algorithms. LNMCS did not outperform NMCS due to the nature of the search space: there rarely is a move that doom all its child-states It seems that not one single algorithm is able to solve all of sample of conjectures we selected, and using different approaches is required to adapt to the variety of problems in spectral graph theory conjecture refutation. We recommend trying GBFS, the NMCS, and then NRPA.

Trying to build trees even when the conjecture is applied on any graph can also be helpful as it reduces the amount of possible builds greatly and focuses on a sub-type of graphs that often features extreme properties over the spectral invariants. It is inexpensive and should be tried first.

However, these methods present limits. Computing score functions that require eigenvalues on big trees (over size 300) can be very costly. They are also dependent on the shape of the score function: a noisy score function with many local minimums can be challenging, as well as a score function with more discrete results can lead to an absence of differentiation in the paths to explore. Conjectures requiring computing a NP-hard problem can also severely increase the computing time even for small graphs (30 vertices).

Despite these limitations, and depending on the definition of the gravity matrix, either our program was able to refute up to the state of the art (manually from graph theorists) the conjectures from Graffiti, or it refuted Graffiti 197, a previously open conjecture.

You can access the implementations of the conjecture and the Rust code used to refute them here:

https://github.com/RoucairolMilo/refutationExperimentalMathematics

Statements and Declarations

Thanks to Édouard Bonnet for fruitful discussions. Thanks to Craig E. Larson and Ermelinda DeLaViña for answering some of our questions and providing "written on the wall" to us.

6.1 funding

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).

6.2 competing interests

The authors have no relevant financial or non-financial interests to disclose.

References

1. M. Aouchiche and P. Hansen. A survey of automated conjectures in spectral graph theory. *Linear Algebra and its Applications*, 432(9):2293–2322, April 2010.
2. Tony L Brewster, Michael J Dinneen, and Vance Faber. A computational attack on the conjectures of graffiti: New counterexamples and proofs. *Discrete Mathematics*, 147(1-3):35–55, 1995.
3. Tristan Cazenave. Nested Monte-Carlo Search. In Craig Boutilier, editor, *IJCAI*, pages 456–461, 2009.
4. Tristan Cazenave. Generalized rapid action value estimation. In *24th International conference on artificial intelligence*, pages 754–760, 2015.
5. Ermelinda Delavina. Some History of the Development of Graffiti. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science 69: Graphs and Discovery*, pages 81–118, 2005.

6. O. Favaron, M. Mahéo, and J.-F. Saclé. Some eigenvalue properties in graphs (conjectures of graffiti — ii). *Discrete Mathematics*, 111(1):197–220, 1993.
7. Odile Favaron, Maryvonne Mahéo, and J-F Saclé. On the residue of a graph. *Journal of Graph Theory*, 15(1):39–64, 1991.
8. Sylvain Gelly and David Silver. Monte-carlo tree search and rapid action value estimation in computer go. *Artificial Intelligence*, 175(11):1856–1875, 2011.
9. Pierre Hansen and Gilles Caporossi. AutoGraphiX: An Automated System for Finding Conjectures in Graph Theory. *Electronic Notes in Discrete Mathematics*, 5:158–161, July 2000.
10. Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *17th European Conference on Machine Learning (ECML'06)*, volume 4212 of *LNCS*, pages 282–293. Springer, 2006.
11. Christopher D. Rosin. Nested rollout policy adaptation for monte carlo tree search. In *In IJCAI*, pages 649–654, 2011.
12. Milo Roucairol and Tristan Cazenave. Refutation of spectral graph theory conjectures with monte carlo search. In *International Computing and Combinatorics Conference*, pages 162–176. Springer, 2022.
13. Milo Roucairol and Tristan Cazenave. Solving the hp model with nested monte carlo search. *arXiv preprint arXiv:2301.09533*, 2023.
14. D. Silver, Aja Huang, Chris J. Maddison, A. Guez, L. Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, S. Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.
15. David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017.
16. Adam Zsolt Wagner. Constructions in combinatorics via neural networks. *arXiv:2104.14516 [cs, math]*, April 2021. arXiv: 2104.14516.

Table 1. Times in seconds to obtain a refutation by applying every algorithm on selected Graffiti conjectures.

Graffiti	Size	Graph type	NMCS	LNMCS	NRPA	UCT	GBFS	BEAM	GRAVE	RAVE
166 R	5+	any	0	0	0	0	0	0	0	0
321 R	4+	$K3-free$	0	0	0	0	0	0	0	0
189 R	5+	any	0	0	0	2	10	4	2	-
289 R	20+	girth ≥ 5	600	600	200	-	6	102	-	-
302 R	6+	tree	0	0	0	0	0	0	0	0
715 R	16+	tree	0	0	0	0	1	0	0	0
29 R	7+	any	2	2	10	5	0	-	2	1
30 R	12+	any	0	0	0	0	311	-	0	0
301 R	14+	tree	2	2	4	-	0	0	7	2
137 R	67+	any	-	-	-	-	513	-	-	-
139 R	50+	any	-	-	-	-	36	-	-	-
711 R	5+	any	0	0	4	0	0	0	0	0
197 O	17+	$K3-free$	30	30	5	-	0	4	-	-
140 R	50	any & tree	-	-	-	-	-	-	-	-
290 O	50	girth ≥ 5	-	-	-	-	-	-	-	-
284 O	50	girth ≥ 5	-	-	-	-	-	-	-	-
195 O	50	any	-	-	-	-	-	-	-	-
21 O	50	any & tree	-	-	-	-	-	-	-	-
39 O	50	any & tree	-	-	-	-	-	-	-	-
143 O	100	any & tree	-	-	-	-	-	-	-	-
154 O	50	any & tree	-	-	-	-	-	-	-	-
20 O	50	any & tree	-	-	-	-	-	-	-	-
40 O	50	any & tree	-	-	-	-	-	-	-	-
254 O	50	any & tree	-	-	-	-	-	-	-	-
262 O	50	any & tree	-	-	-	-	-	-	-	-
712 O	50	any & tree	-	-	-	-	-	-	-	-
198 O	50	any	-	-	-	-	-	-	-	-
714 O	100	any	-	-	-	-	-	-	-	-
219 O	50	$K3-free$	-	-	-	-	-	-	-	-
322 O	50	$K3-free$	-	-	-	-	-	-	-	-
292 O	50	girth ≥ 5	-	-	-	-	-	-	-	-
295 O	50	girth ≥ 5	-	-	-	-	-	-	-	-
129 O	50	any	-	-	-	-	-	-	-	-
698 O	50	any	-	-	-	-	-	-	-	-
252 O	50	any	-	-	-	-	-	-	-	-

Graffiti: R means the conjecture was already refuted (as reported in [1]) O means the conjecture is open to be proved or refuted
Size: If there is a "+", it is the size at which we start to find counter-examples with our algorithms. Otherwise, it is the size we built graphs up to.
Graph type: It is the types of graphs we used in our searches, "any" means there was no restriction on the graph built, $K3-free$ means there is no clique of size 3, "girth ≥ 5" means the shortest cycle must be longer than 5, "tree" means the graph must be a tree. We tried both generating trees and then any graph on multiple conjectures.
A 0 in the time section means that the conjecture was refuted in less than a second.

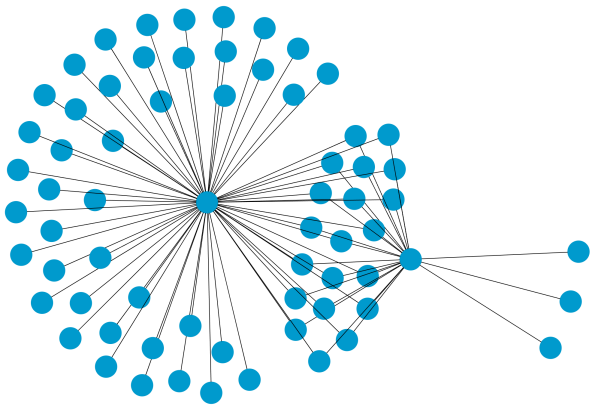


Fig. 5. A counter-example of Graffiti 137 of size 67 (second largest eigenvalue $\leq$ harmonic)
Edges: 0-1, 0-2, 1-3, 1-4, 1-5, 1-11, 1-15, 1-18, 1-21, 1-25, 1-28, 1-31, 1-34, 1-37, 1-40, 1-43, 1-46, 1-49, 1-52, 1-55, 1-58, 1-61, 1-64, 1-66, 2-6, 2-7, 2-8, 2-9, 2-10, 2-11, 2-12, 2-13, 2-14, 2-15, 2-16, 2-17, 2-18, 2-19, 2-20, 2-21, 2-22, 2-23, 2-24, 2-25, 2-26, 2-27, 2-28, 2-29, 2-30, 2-31, 2-32, 2-33, 2-34, 2-35, 2-36, 2-37, 2-38, 2-39, 2-40, 2-41, 2-42, 2-43, 2-44, 2-45, 2-46, 2-47, 2-48, 2-49, 2-50, 2-51, 2-52, 2-53, 2-54, 2-55, 2-56, 2-57, 2-58, 2-59, 2-60, 2-61, 2-62, 2-63, 2-64, 2-65, 2-66

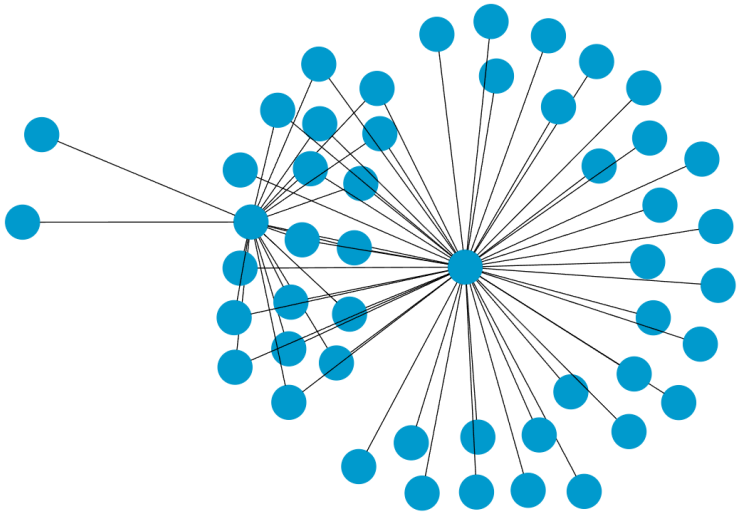


Fig. 6. A counter-example of Graffiti 139 of size 50 (- second smallest eigenvalue $\leq$ harmonic)
Edges: 0-1, 0-2, 0-3, 0-4, 0-5, 0-8, 0-11, 0-14, 0-17, 0-20, 0-23, 0-26, 0-29, 0-32, 0-35, 0-38, 0-41, 0-44, 0-47, 0-49, 1-2, 2-3, 2-6, 2-7, 2-8, 2-9, 2-10, 2-11, 2-12, 2-13, 2-14, 2-15, 2-16, 2-17, 2-18, 2-19, 2-20, 2-21, 2-22, 2-23, 2-24, 2-25, 2-26, 2-27, 2-28, 2-29, 2-30, 2-31, 2-32, 2-33, 2-34, 2-35, 2-36, 2-37, 2-38, 2-39, 2-40, 2-41, 2-42, 2-43, 2-44, 2-45, 2-46, 2-47, 2-48, 2-49

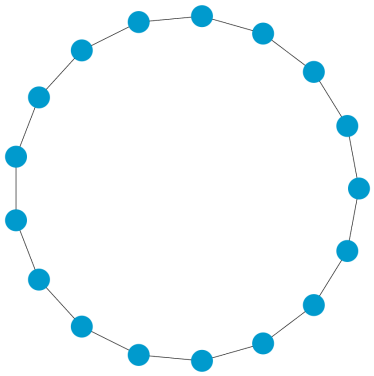


Fig. 7. A counter-example of Graffiti 197 of size 17
Edges: 0-1, 1-2, 02-3, 3-4, 4-5, 5-6, 6-7, 7-8, 8-9, 9-10, 10-11, 11-12, 12-13, 13-14, 14-15, 15-16, 16-0

ORIGINAL RESEARCH — WORD COUNT : 3506

Automated Refutation with Monte Carlo Search of Graph Theory Conjectures on the Maximum Laplacian Eigenvalue

Liora Taieb[a], Tristan Cazenave[a], Milo Roucairol[a] and Ararat Harutyunyan[a]

[a]LAMSADE, Université Paris Dauphine - PSL

ARTICLE HISTORY
Compiled October 4, 2024

ABSTRACT
We address the problem of automatic refutation of spectral graph theory conjectures with Monte Carlo methods. Usual ways are testing conjectures on an exhaustive database of graphs below a certain size, local search algorithms, or, more recently, deep reinforcement learning. We expand on previous works by finding smaller (and often sparser) counter-examples to spectral graph theory conjectures in seconds when it takes minutes or hours with other methods. We apply search algorithms (including state-of-the-art Monte Carlo Searches) to 68 automated conjectures already addressed by the deep cross-entropy method. In addition to the ones already disproved by deep cross-entropy, we refute 2 open conjectures until now. We highlight the efficiency of Monte Carlo Search algorithms compared to a state-of-the-art neural approach, and the advantages of the constructive method. Monte Carlo search can be used to automatically refute conjectures that are experimentally generated.

KEYWORDS
Monte Carlo Search; Spectral Graph Theory; Conjecture; Refutation

1. Introduction

Finding potential counterexamples to a graph theory conjecture can be a tiresome task. The search space of graphs of order n consists of at least of $\frac{2^{\binom{n}{2}}}{n!}$ non isomorphic graphs, with $n!$ being the maximal cardinality of a graph isomorphism class. Thus the search space's size is expanded in doubly exponential fashion when exploring multiple graph sizes when there is no prior intuition on the likely size of a counter-example.

This article expands the work of Roucairol and Cazenave (Roucairol & Cazenave, 2022, 2024), who were the first to use Monte Carlo Search methods to build counter-examples of spectral graph theory conjectures, in which they showed the effectiveness of the approach against neural state-of-the-art methods. Few works have been devoted to evaluating Monte Carlo search algorithms for combinatorial graph problems. Cazenave et al. applied it to graph coloring (Cazenave, Negrevergne, & Sikora, 2021), investigated before them only by Edelkamp et al. (Edelkamp, Externest, Kühl, & Kuske, 2017), both works being competitive with state-of-the-art SAT solvers.

Our focus in this paper is once again on spectral graph theory conjectures, as they are well-suited for automated refutation. Here, we focus on conjectures involving the maximum Laplacian eigenvalue of a graph. The property of a spectral theory conjecture

CONTACT Author. Email: tristan.cazenave@lamsade.dauphine.fr

can straightforwardly be translated into an evaluation function whose set of images is large and continuous, which usually provides a good assessment of the quality of the states evaluated. Many available softwares and libraries of various computer languages carry out eigenvalue calculations quickly.

The paper is organized as follows. In Sect. 2, we describe previous work in the refutation of graph theory conjectures and we focus on Monte Carlo search. In Sect. 3, we present the problem and the methodology used to explore the problem space. Finally, in Sect. 4 and 5, we present and discuss our results on multiple conjectures.

2. Refutation of Graph Theory Conjectures

2.1. State Of The Art

Graph conjectures are propositions on graph classes (any graph, trees, K-free...) that are thought to be true and are awaiting proof or a refutation. Many mathematicians have put forward spectral graph theory conjectures still open to this day (Liu & Ning, 2023). Automated softwares for creating conjectures emerged at the rise of powerful computers in the 80s, including Ingrid (Dutton, Brigham, & Gomez, 1989), GRAPH (Cvetković & Simić, 1994), Graffiti (DeLaVina, 2005) and AutoGraphiX (P. Hansen & Caporossi, 2000). Thanks to them, plenty of such conjectures are available and still open (Aouchiche & Hansen, 2010).

Graffiti uses known theorems to create and refine many conjectures in the form of inequalities between graph invariants. It then tests these conjectures on a database of graphs and discards the falsified ones. Its database is built by an exhaustive generation of graphs smaller than a threshold size. The system later checks if the inequalities are not implied by already known theorems and conjectures, including those it has created itself. If a conjecture passes these tests, it is proposed to graph theorists. "Written on the wall" (*Latest version of "Written on the wall"*, 2012) collects almost 1000 conjectures from Graffiti, along with the discussions of many renowned graph theorists. The efficiency of Graffiti's refutation process is strongly limited by the quality of the database, in other words its completeness and the maximal size of generated graphs.

AutoGraphiX uses local search to create - and also refute - some conjectures. The program uses a heuristic, the Variable Neighborhood Search (VNS) to identify extremal graphs and suggest conjectures based on their structure. Local search is naturally more intelligent than a naive exhaustive generation as it introduces constraints to reduce the search space, and has been used several times to solve combinatorial graph problems (Hertz & de Werra, 1987; Lidický, McKinley, & Pfender, 2024; Mehrabian et al., 2023).

Lately, machine and deep learning tackled combinatorial problems over graphs (Khalil, Dai, Zhang, Dilkina, & Song, 2017). Wagner's deep reinforcement learning technique (Wagner, 2021) is a pioneering refutation method of graph theory conjectures. It has been reworked in (Angileri et al., 2024) and used for research in Turán theory in (Mehrabian et al., 2023), Ramsey theory in (Ghebleh, Al-Yakoob, Kanso, & Stevanović, 2024) and spectral graph theory in (Al-Yakoob, Ghebleh, Kanso, & Stevanovic, 2024), disproving open conjectures in (Al-Yakoob et al., 2024; Ghebleh et al., 2024; Wagner, 2021). This program learns a policy of graph generation using a deep cross entropy. It trains a neural network to output a policy on the next edge to add to a graph. This way, it creates a batch of full graphs from which the network learns by minimizing the cross-entropy with the distribution of the best graphs of the batch. Wagner's method intrinsically understands the structure of the best graphs. But it

can take hours or days, and it is limited to the study of fixed sizes of graphs.

2.2. Monte Carlo Search

Monte Carlo Tree Search (MCTS) is a family of algorithms that combine tree search and reinforcement learning using playouts and heuristics (Browne et al., 2012). The search space (and not the search states) is represented by a tree, where the nodes are the partial constructions, the edges are the next possible moves and the leaves are terminal states. It performs numerous simulations and stores the statistics of actions and their resulting evaluations to make more educated choices in each iteration. Constraints on legal moves delimit the possible extent of the search space.

Monte Carlo search algorithms were proven to be powerful in puzzles and optimization problems, with recent successes like AlphaGo (Silver et al., 2016). They often excel in games (Cazenave, 2009; Méhat & Cazenave, 2010) and non-games applications including biology (Edelkamp & Tang, 2015; Portela, 2018), logistics (Edelkamp et al., 2016; Edelkamp & Greulich, 2014) and many more (Browne et al., 2012). These algorithms have the advantage of only needing an evaluation function for the final state of the space they explore.

The efficiency of MCTS comes from its incremental construction of graphs, which can produce large solutions. It is similar to local search in that it improves the construction by moving towards the best neighboring solutions. As a local search and reinforcement learning method, MCTS can fall into a local optima if the evaluation score is too noisy, but decisions based on reinforcement learning help balance this limitation. Some of them such as Nested Rollout Policy Adaptation learn a general policy that serves as a playout guiding heuristic, in line with the principle of the deep cross entropy, without being limited by a fixed graph size.

MCTS is efficient in solving optimization problems where the score function and the legal moves are relatively inexpensive to compute, as these calculations will be made many times during the simulations. The difficulty with these methods lies in the choice of an efficient exploration of the search space, and the formalization of an objective yet feasible evaluation function. For NP-hard problems, it is possible to bias the search intelligently (Cazenave, 2017; Cazenave et al., 2021).

3. Problem and Methodology

The conjectures we study come from (Brankov, Hansen, & Stevanović, 2006) and involve the largest eigenvalue μ of the Laplacian matrix of graphs (degree matrix minus adjacency matrix). They were created automatically similarly to Graffiti with a database of 273 214 connected graphs with up to 9 vertices, enhanced by a few special graphs. We focus on the 68 conjectures attempted by Al-Yakoob et al. with the deep cross entropy method (Al-Yakoob et al., 2024). All conjectures are presented in the appendix, their order comes from (Al-Yakoob et al., 2024), different from the order (Brankov et al., 2006). They are of the form

$$\mu \leqslant \max_{v_i} f\left(d_i, m_i\right) \quad (1) \qquad \text{or} \qquad \mu \leqslant \max_{v_i \sim v_j} f\left(d_i, m_i, d_j, m_j\right) \quad (2)$$

where v_i is any vertex of G, d_i is the degree of this vertex and m_i is the average of the degrees of v_i's neighbors, and f is some function involving these parameters.

The evaluation function naturally is μ - the value on the right of the inequality.

To avoid floating point errors when calculating eigenvalues, we force the result to be greater than 0.0001 for the graph to be considered a counter-example (it is more than enough for graphs of size 20). The conjectures apply to any graph, therefore we begin with the naive approach of not restraining the legal moves. Following the work of Roucairol and Cazenave (Roucairol & Cazenave, 2022), we then narrow the legal moves to create only trees, as the search space is way smaller and trees are more likely to converge toward extreme graphs with extreme properties.

We apply 3 Monte Carlo search algorithms to the problem as well as the Greedy Best-First Search algorithm, the evolutionary algorithm Covariance Matrix Adaptation - Evolutionary Strategy and an Iterated Local Search. Their pseudo-codes are presented in the appendix.

- Nested Monte Carlo Search (NMCS) (Cazenave, 2009) uses nested levels of playouts with random playouts at the base level. At each recursion level, each legal moves is assigned a score from the results of the lower level NMCS starting from the move, and the best move is selected this way.
- Nested Rollout Policy Adaptation (NRPA) (Rosin, 2011) is similar to a NMCS, but learns a policy with nested levels of best sequences. At the lowest level, it becomes a playout with the learned policy.
- Generalized Rapid Action Value Estimation (GRAVE) (Cazenave, 2015) uses the All Moves As First (AMAF) heuristic to update move statistics, taking into account all the moves that were played in the playout and not only the first one. It incorporates statistics from a higher reference state in the tree, which is the closest ancestor state that has more playouts than a given constant.
- Greedy Best-First Search (GBFS) is a simple and deterministic greedy algorithm opening the best state from a list, evaluating the children of this state, and inserting these children back in the list according to their evaluation.
- Covariance Matrix Adaptation - Evolutionary Strategy (CMA-ES) (N. Hansen, 2016) is an evolutionnary method that samples children from the multivariate Gaussian distribution of the best parents, with the aim of producing even better children. By definition, it understands the structures of best graphs to which it applies mutations.
- Iterated Local Search (ILS) (Lourenço, Martin, & Stützle, 2019) repeatedly applies local search on perturbed solutions.

4. Results

The data and code that support the findings of this study are openly available at:
https://github.com/liorataieb/RefutationSpectralConjectures.

The experiments were made with Rust 2024.1, on an Intel Core i7-1365U 5.2GHz using a single core. Each algorithm has been allocated a maximum of 1 minute per conjecture and each algorithm has been ran several times. The then unrefuted conjectures have been alloted additional time, between 15 minutes and 1 hour. The terminal state of the algorithms and playouts occurs when the graph reaches 20 vertices, as it is the fixed size studied with the deep cross entropy. We also tried various terminal sizes between 15 and 50. We present in Table 1 the results of the algorithms, only for the refuted conjectures for the sake of clarity.

- NCMS, NRPA and GRAVE were ran with and without heuristic on the choice of the next move in playouts.

- Only NMCS, NRPA, GRAVE and GBFS were restricted to trees.
- NMCS and NRPA used at most a level of 3.
- GRAVE used a reference of 50.
- CMA-ES used a λ of 5, 10 or 15.
- Several variations of ILS were tried, inspired from the work in (Lourenço et al., 2019).
- CMA-ES and ILS were tried on solutions of size 20.

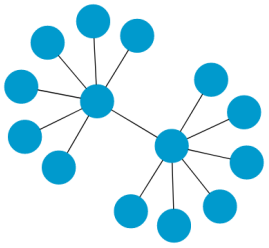
The deep cross entropy in (Al-Yakoob et al., 2024) disproved 25 conjectures mostly with graphs of size 20, up to size 24. 5 more conjectures were refuted in the article using an exhaustive research on subquartic graphs (graph with degree at most 4) of size 14 at most. As shown in Table 1, our algorithms combined were able to refute 29 conjectures : the 25 refuted by the deep cross entropy, 2 that were still open (Conjectures 45 and 48, see Figure 1a and Figure 1b) and 2 (out of 5) that were refuted by a subquartic graph.

By aggregating the results of 1-minute runs, NMCS outperforms the deep cross entropy and refutes 28 conjectures, NRPA refutes 26, GRAVE refutes 26 and GBFS refutes 16. ILS and CMA-ES refute no conjectures. The best number of disproved conjectures obtained without restart, leaving only 1 minute by conjecture, is 23 conjectures for NMCS, NRPA and GRAVE, achieved when we restrict the search space to trees and use a level of 3 for NMCS and NRPA. with the same constraints applied to exploration of the entire search space, NMCS and GRAVE produced a maximum of 17 refuted conjectures, NRPA produced at most 4.

Parallelized restarts of MCTS methods are strongly advised given the difference between the total number of conjectures refuted and the one of the best run of each algorithm. Our experiments show that applying a heuristic to moves during the playouts systematically increases the number of refuted conjectures compared with non-heuristic applications of the algorithms, although the refuted conjectures are not always the same. The same goes when searching for trees rather than any graphs. It is truly a cost-effective variation that should be tried first.

Increasing the level of NRPA and NMCS yields more refutation, but not every conjectures can be refuted with a higher level. Conjecture 28 was refuted by NRPA only with a level of 2 or lower. Likewise, Conjecture 48 can be disproved by NMCS only with a level of 2 or lower, because NMCS with a level of 3 starts evaluating graphs at size 4. All other conjectures have been refuted with a level of 3. We do not advise to go higher than a level of 3, as the trade-of between detailed exploration and computational power requirements is profitable. As for the terminal size of graphs, trying different ones is good practice; most of the conjectures were refuted with a size of 20, but not all, as Conjecture 62 with GRAVE was refuted only with a size of 50 when restricting the search space to trees.

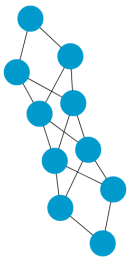
It is interesting to note that GRAVE is the only algorithm that found a counter-example for Conjecture 51. NRPA favors patterns and NMCS succeeds more where the solutions are more chaotic. GRAVE can be described as a mix of the two (not in terms of algorithm, but policy learning behavior), and it is illustrated by the structure of the counter-example found (see Figure 2b). Furthermore, NMCS yields smaller counter-examples than NRPA, itself producing smaller results than GRAVE. NRPA and GRAVE execute a repetitive strategy usually leading to larger graphs. Given that the counter-examples that are not trees found by NMCS and GRAVE have very little structure, 1 minute may not be enough for NRPA to produce solutions, explaining the discrepancy in results between the algorithms when they are not limited to the search



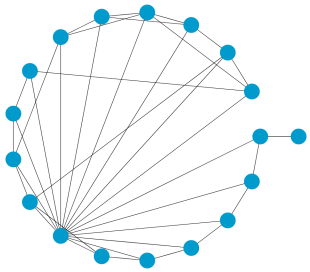
(a) Edges : 0-1, 0-4, 0-5, 0-8, 0-10, 0-11, 0-12, 1-2, 1-3, 1-6, 1-7, 1-9, 1-13

(b) Edges : 0-1, 0-2

Figure 1. A counter-example of Conjecture 45 (left) and Conjectures 48, 57 and 61 (right)



(a) Edges : 0-1, 0-9, 1-2, 1-8, 2-3, 2-7, 2-9, 3-4, 3-6, 3-8, 4-5, 4-7, 5-6, 6-7, 7-8, 8-9

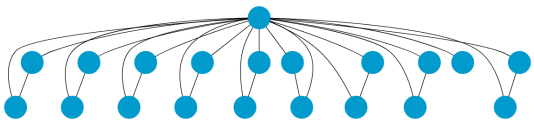


(b) Edges : 0-1, 0-2, 0-3, 0-4, 0-5, 0-6, 0-7, 0-8, 0-9, 0-10, 0-11, 0-12, 0-14, 0-15, 0-16, 0-17, 1-2, 1-9, 1-17, 2-3, 2-5, 3-4, 4-10, 5-6, 5-7, 6-7, 6-8, 7-8, 7-10, 8-9, 9-10, 11-12, 11-13, 12-14, 14-15, 15-16, 16-17

Figure 2. A counter-example of Conjecture 50 (left) and Conjecture 51 (right)

for trees.

Among the 11 conjectures disproved by MCTS but not GBFS, the latter is only able to refute 1 more conjecture with a 5-minute run, and 3 more with 15 minutes of allotted time. Conjecture 50 was refuted only by NMCS during a 1-hour run with a terminal size of 15, raising the total number of conjectures disproved by the algorithm to 28. As shown in Figure 2a, the counter-example is very structured but small, perhaps too small for NRPA and GRAVE to find it. We tried several 1-hour runs of NMCS with heuristic, NRPA and GBFS, both for any graphs and for trees, on the 3 remaining conjectures already refuted by subquartic graphs, without results. Moreover, NMCS, NRPA, GRAVE, and GBFS were not able to refute unproved conjectures with a 15-minute alloted time on terminal size 15, 20 and 50. However the bounds of those conjectures are often reached during exploration.



(a) Edges : 0-1, 0-2, 0-3, 0-4, 0-5, 0-6, 0-7, 0-8, 0-9, 0-10, 0-11, 0-12, 0-13, 0-14, 0-15, 0-16, 0-17, 0-18, 0-19, 2-3, 4-5, 6-7, 8-9, 10-11, 12-13, 14-15, 16-17, 18-19

Figure 3. A counter-example of Conjecture 65

Table 1. Time in seconds to obtain a refutation with each algorithm (maximum of 1 minute per conjecture).

	NMCS		NRPA		GRAVE		GBFS		CMA-ES	ILS
Conj.	Time	Graph type	Time	G. type	Time	G. type	Time	G. type		
3	0	tree	50	tree	0	tree	0	any	-	-
15	0	tree	0	tree	7	tree	1	any	-	-
28	0	tree	0	tree	-	-	0	any	-	-
29	0	tree	0	tree	0	tree	-	-	-	-
31	0	tree	1	tree	0	tree	25	any	-	-
36	0	tree	0	tree	0	tree	-	-	-	-
	43	any								
41	3	any	14	tree	7	any	-	-	-	-
43	0	tree	0	tree	0	tree	0	any	-	-
	2	any			1	any				
45	0	tree	3	tree	-	-	-	-	-	-
48	0	tree	0	tree	0	tree	0	tree	-	-
	0	any	0	any	0	any	0	any		
49	0	tree	0	tree	0	tree	0	any	-	-
	7	any			3	any				
50	422[b]	any	-	-	-	-	-	-	-	-
51	-	-	-	-	5	any	-	-	-	-
52	0	tree	0	tree	0	tree	0	any	-	-
	1	any			0	any				
53	0	tree	0	tree	0	tree	0	any	-	-
	1	any			0	any				
54	0	tree	0	tree	0	tree	-	-	-	-
	1	any			0	any				
55	0	tree	0	tree	0	tree	-	-	-	-
	3	any			0	any				
57	0	tree	0	tree	0	tree	0	tree	-	-
	1	any	0	any	0	any	0	any		
58	0	tree	0	tree	0	tree	0	any	-	-
	3	any	43	any	4	any				
59	1	tree	0	tree	0	tree	-	-	-	-
	11	any			19	any				
60	1	tree	2	tree	0	tree	-	-	-	-
	1	any			25	any				
61	0	tree	0	tree	0	tree	0	tree	-	-
	1	any	0	any	0	any	0	any		
62	1	tree	0	tree	0	tree	-	-	-	-
	2	any			4	any				
63	0	tree	0	tree	0	tree	0	any	-	-
	23	any								
64	1	tree	2	tree	0	tree	0	any	-	-
65	0	tree	0	tree	0	tree	-	-	-	-
	1	any			0	any				
66	48	tree	-	-	1	tree	0	any	-	-
67	1	tree	3	tree	0	tree	-	-	-	-
68	0	tree	0	tree	0	tree	0	any	-	-
	1	any	4	any	0	any				

[a]types of graphs we used in our searches, "any" means there was no restriction on the graph built, "tree" means the graph must be a tree. [b]Those results were obtained by NMCS while testing 1-hour runs per conjecture for Conjecture 2, 17, 32 and 50, the 4 conjectures refuted by subquartic graphs but not MCTS.

5. Discussion

All conjectures previously refuted with the deep cross entropy have been disproved very quickly by the Monte Carlo algorithms, especially NMCS and GRAVE. The difference of results between MCTS and GBFS highlights MCTS superior exploration capabilities. This study provides evidence that MCTS discovers potentially desirable structures much sooner than deep neural reinforcement learning and outperforms both the deep cross entropy and more naive local searches. MCTS strength comes from the gradual building of graphs coupled with smart detection of a search branch quality. It gives them an advantage to find counter-examples of any (reasonable) sizes, especially smaller and typically less dense counter-examples. An extreme illustration of this is Figure 1b, the very small path of 3 vertices, that refutes Conjectures 48, 57 and 61, one of which was still open. The simplicity of the counter-example exhibits a serious limitation of the exhaustive generation used to test to conjectures in (Brankov et al., 2006). But it is a very simple graph, directly and naturally found by MCTS algorithms.

The 4 subquartic graphs presented in (Al-Yakoob et al., 2024) that were able to refute Conjectures 2, 17, and 32 are very specific. MCTS is capable of generating specific graphs, such as the one in Figure 3a, which presents an almost windmill[1] graph as a counter-example to Conjecture 36, and the one in Figure 2a, which presents a "shoelace" graph as a counter-example to Conjecture 50. The fact that MCTS failed to produce counter-examples for Conjectures 2, 17, and 32 even after hour-long runs highlights the lack of guarantee in finding an optimal result. The bound of those conjectures are reached by our algorithms, but we believe this limitation is due to the high granularity of the score functions. We tried to remove arbitrarily one or two edges of the subquartic counter-examples in (Al-Yakoob et al., 2024) and spotted significantly lower scores when doing so, so much that no conjecture was refuted anymore. This is a general limitation of optimization methods.

CMA-ES and ILS have not been restricted to build trees, which may explain the poor results we obtained for them. Both methods start with a preconstructed graph. CMA-ES is the closest algorithm to the deep cross entropy, and also the hardest to train. We believe it got lost in the granularity of the score functions. ILS seems too naive to yield results.

6. Conclusion

The Monte Carlo approach performs well in many fields, and their versatility can effectively be applied to conjecture refutation in graph theory as well. We have shown that Monte Carlo search rapidly outperforms its rivals in refuting spectral graph theory conjectures. Its intelligence allows for quick and effortless research among many sizes of graphs. Roucairol and Cazenave (Roucairol & Cazenave, 2022) have shown that Monte Carlo methods can rapidly produce large counter-examples. Here, the methods yield smaller counter-examples than the state-of-the-art deep cross entropy method. It also refutes open conjectures which were not refuted by other methods.

Here, we only used Monte Carlo methods, as the conjectures we studied call for computationally inexpensive calculations, still on a par with what is done today. These methods are adaptable and can very well be combined with diverse heuristics to boost

[1] A windmill is an undirected graph constructed for $k \leqslant 2$ and $n \leqslant 2$ by joining n copies of the complete graph K_k at a shared universal vertex.

results. Monte Carlo search should be investigated further for more complex combinatorial graph problems.

Disclosure statement

The authors have no relevant financial or non-financial interests to disclose.

Funding

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).

7. References

References

Al-Yakoob, S., Ghebleh, M., Kanso, A., & Stevanovic, D. (2024). Reinforcement learning for graph theory, i. reimplementation of wagner's approach. *arXiv preprint arXiv:2403.18429*.

Angileri, F., Lombardi, G., Fois, A., Faraone, R., Metta, C., Salvi, M., & Morandin, F. (2024). A systematization of the wagner framework: Graph theory conjectures and reinforcement learning. *arXiv preprint arXiv:2406.12667*.

Aouchiche, M., & Hansen, P. (2010). A survey of automated conjectures in spectral graph theory. *Linear algebra and its applications*, *432*(9), 2293–2322.

Brankov, V., Hansen, P., & Stevanović, D. (2006). Automated conjectures on upper bounds for the largest laplacian eigenvalue of graphs. *Linear algebra and its applications*, *414*(2-3), 407–424.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., & Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, *4*(1), 1–43.

Cazenave, T. (2009, June). Nested monte-carlo search. In *Twenty-first international joint conference on artificial intelligence.*

Cazenave, T. (2015). Generalized rapid action value estimation. In *24th international conference on artificial intelligence* (pp. 754–760).

Cazenave, T. (2017). Nested rollout policy adaptation with selective policies. In T. Cazenave, M. H. M. Winands, S. Edelkamp, S. Schiffel, M. Thielscher, & J. Togelius (Eds.), *Cgw/giga-2016* (Vol. 705, pp. 44–56). Springer, Cham. Retrieved from `https://doi.org/10.1007/978-3-319-57969-6`$_4$

Cazenave, T., Negrevergne, B., & Sikora, F. (2021). Monte carlo graph coloring. In *Monte carlo search: First workshop, mcs 2020, held in conjunction with ijcai 2020, virtual event, january 7, 2021, proceedings 1* (pp. 100–115). Springer International Publishing.

Cvetković, D., & Simić, S. (1994). Graph theoretical results obtained by the support of the expert system" graph". *Bulletin (Académie serbe des sciences et des arts. Classe des sciences mathématiques et naturelles. Sciences mathématiques)*, 19–41.

DeLaVina, E. (2005). Some history of the development of graffiti. In *Dimacs series in discrete mathematics and theoretical computer science* (Vol. 69, p. 81).

Dutton, R. D., Brigham, R. C., & Gomez, F. (1989). Ingrid: A graph invariant manipulator. *Journal of symbolic computation*, *7*(2), 163–177.

Edelkamp, S., Externest, E., Kühl, S., & Kuske, S. (2017). Solving graph optimization problems in a framework for monte-carlo search. In *Tenth annual symposium on combinatorial search.*

Edelkamp, S., Gath, M., Greulich, C., Humann, M., Herzog, O., & Lawo, M. (2016). Monte-carlo tree search for logistics. In *Commercial transport: Proceedings of the 2nd interdisciplinary conference on production logistics and traffic 2015* (pp. 427–440). Springer International Publishing.

Edelkamp, S., & Greulich, C. (2014). Solving physical traveling salesman problems with policy adaptation. In *2014 ieee conference on computational intelligence and games* (pp. 1–8).

Edelkamp, S., & Tang, Z. (2015). Monte-carlo tree search for the multiple sequence alignment problem. In *Proceedings of the international symposium on combinatorial search* (Vol. 6, pp. 9–17).

Ghebleh, M., Al-Yakoob, S., Kanso, A., & Stevanović, D. (2024). Reinforcement learning for graph theory, ii. small ramsey numbers. *arXiv preprint arXiv:2403.20055*.

Hansen, N. (2016). The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*.

Hansen, P., & Caporossi, G. (2000). Autographix: An automated system for finding conjectures in graph theory. *Electronic Notes in Discrete Mathematics*, *5*, 158–161.

Hertz, A., & de Werra, D. (1987). Using tabu search techniques for graph coloring. *Computing*, *39*(4), 345–351.

Khalil, E., Dai, H., Zhang, Y., Dilkina, B., & Song, L. (2017). Learning combinatorial optimization algorithms over graphs. *Advances in Neural Information Processing Systems*, *30*.

Latest version of "written on the wall". (2012). (Accessed: https://independencenumber.wordpress.com/wp-content/uploads/2012/08/wow-july2004.pdf)

Lidický, B., McKinley, G., & Pfender, F. (2024). Small ramsey numbers for books, wheels, and generalizations. *arXiv preprint arXiv:2407.07285*.

Liu, L., & Ning, B. (2023). Unsolved problems in spectral graph theory. *arXiv preprint arXiv:2305.10290*.

Lourenço, H. R., Martin, O. C., & Stützle, T. (2019). Iterated local search: Framework and applications. In *Handbook of metaheuristics* (pp. 129–168).

Mehrabian, A., Anand, A., Kim, H., Sonnerat, N., Balog, M., Comanici, G., & Wagner, A. Z. (2023). Finding increasingly large extremal graphs with alphazero and tabu search. *arXiv preprint arXiv:2311.03583*.

Méhat, J., & Cazenave, T. (2010). Combining uct and nested monte carlo search for single-player general game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, *2*(4), 271–277.

Portela, F. (2018). An unexpectedly effective monte carlo technique for the rna inverse folding problem. *BioRxiv*, 345587.

Rosin, C. D. (2011, July). Nested rollout policy adaptation for monte carlo tree search. In *Ijcai* (Vol. 2011, pp. 649–654).

Roucairol, M., & Cazenave, T. (2022). Refutation of spectral graph theory conjectures with monte carlo search. In *International computing and combinatorics conference* (pp. 162–176). Springer International Publishing.

Roucairol, M., & Cazenave, T. (2024). *Refutation of spectral graph theory conjectures with search algorithms).* Retrieved from `https://arxiv.org/abs/2409.18626`

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., & Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, *529*(7587), 484–489.

Wagner, A. Z. (2021). Constructions in combinatorics via neural networks. *arXiv preprint arXiv:2104.14516*.

8. Appendices

Appendix A. Algorithms

Algorithm 1 The NMCS algorithm.

```
function NMCS(current-state, level)
    if level = 0 then
        ply ← 0
        seq ← {}
        while current-state is not terminal do
            move ← randomChoice(LegalMoves_current-state)
            current-state ← play(current-state, move)
            seq[ply] ← move
            ply+ = 1
        end while
        return score(current-state), seq
    else
        best-score ← −∞
        best-seq ← []
        ply ← 0
        while current-state is not terminal do
            for each move in M_current-state do
                next-state ← play(current-state, move)
                (score, seq) ←NMCS(next-state, level − 1)
                if score ≥ best-score then
                    best-score ← score
                    best-sequence[ply..] ← move + seq
                end if
            end for
            next-move ← best-sequence[ply]
            ply ← ply + 1
            current-state ← play(current-state, next-move)
        end while
        return (best-score, best-sequence)
    end if
end function
```

Algorithm 2 The NRPA algorithm.

function NRPA(*policy*, *level*)
  if $level = 0$ **then**
    $current\text{-}state \leftarrow root()$
    $ply \leftarrow 0$
    $seq \leftarrow \{\}$
    while *current-state* is not terminal **do**
      $move \leftarrow softmaxChoice(LegalMoves_{current\text{-}state}, policy)$
      $current\text{-}state \leftarrow play(current\text{-}state, move)$
      $seq[ply] \leftarrow move$
      $ply+ = 1$
    end while
    return score $(current\text{-}state, seq)$
  else
    $best\text{-}score \leftarrow -\infty$
    for N *iterations* **do**
      $(result, new) \leftarrow$ NRPA(*policy*, $level - 1$)
      if $result \geq best\text{-}score$ **then**
        $best\text{-}score \leftarrow result$
        $seq \leftarrow new$
      end if
      $pol \leftarrow$ ADAPT(pol, seq)
    end for
    return $(best\text{-}score, seq)$
  end if
end function

function ADAPT(*policy*, *level*)
  $node \leftarrow root()$
  $pol' \leftarrow pol$
  for $ply = 0$ TO $seq - 1$ **do**
    $pol'[(node, seq[ply])] \mathrel{+}= Alpha$
    $z \leftarrow Sum([exp(pol[(node, m)])$ for m in $M_{node}])$
    for each $move$ in M_{node} **do**
      $pol'[(node, move)] \mathrel{-}= \frac{Alpha \cdot exp(pol[(node,move)])}{z}$
    end for
    $node \leftarrow play(node, seq[ply])$
  end for
return pol'
end function

Algorithm 3 The GRAVE algorithm.

Input: N tree-walks, initial state s_0, reference state constant ref
Output: A search tree

Initialize an empty transposition table
for $i = 1$ to N **do**
  $s \leftarrow s_0$, $S \leftarrow \{s\}$, $s_{\text{ref}} \leftarrow s$
  while s is not a leaf state and is not simulatable **do**
    if $n(s) >$ ref **then**
      $s_{\text{ref}} \leftarrow s$
    end if
    for each $a \in s$.children **do**
      $\beta \leftarrow \frac{\text{sref.pAMAF}}{\text{sref.pAMAF}+s.p+\text{bias}\times sref.pAMAF\times s.p}$
      grave $\leftarrow (1-\beta) \times s.\text{mean} + \beta \times sref.\text{AMAF}$
    end for
    Select $a \leftarrow \text{argmax}\{\text{GRAVE}(s,a) \mid a \in s.\text{children}\}$
    Transition to the new state resulting from action a, $S \leftarrow S \cup \{s\}$
  end while
  Sample a new action a from the available moves of s
  Add the state resulting from action a as a child node of s
  while s is not a terminal state **do**
    Sample a from the available moves of s based on the default policy
    Transition to the new state resulting from action a
  end while
  score $\leftarrow$ evaluate(s)
  for each $s \in S$ **do**
    Update s with score
  end for
end for

Algorithm 4 The GBFS algorithm.

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

Algorithm 5 The CMA-ES algorithm.

Initialization:
Number of generations N, number of parents λ.
for each graph size n **do**
    Load graphs from previous iterations (curriculum) and select the best parents.
    if the number of parents is insufficient **then**
        Create additional parents randomly.
    end if
    while N is not reached **do**
        Encode the graphs of the parents as vectors.
        Calculate the mean μ and the covariance matriX Σ of the encoded vectors.
        Write $\Sigma = A^\top A$ using SVD.
        Generate vectors $Y = AX + \mu$ where $X \sim \mathcal{N}(0, I_{n^2})$.
        Construct the graphs of the children from the vectors Y.
        for each generated child **do**
            Calculate the score of the child.
            if the score of the child is higher than the previous best score **then**
                Update the best graphs and the best score.
            else if the score of the child is equal to the previous best score **then**
                if the child is not isomorphic to a previously found graph **then**
                    Add the child to the best graphs.
                end if
            end if
        end for
        Select the best candidates among the parents and children to form the next generation.
    end while
    Increment n, record the best scores and graphs.
end for

Appendix B. Conjectures

The upper bounds of the 68 conjectures that were attempted in this paper. The largest eigenvalue μ of the laplacian matrix of some graph must exceed the bound to refute the conjecture. O means the conjecture is still open to this day, X means it has been refuted.

1. O $\max_{v\in V} \sqrt{\frac{4d_v^S}{m_v}}$
2. X $\max_{v\in V} \frac{2m_v^2}{d_v}$
3. X $\max_{v\in V} \frac{m_v^2}{d_v} + m_v$
4. O $\max_{v\in V} \frac{2d_v^2}{m_v}$
5. O $\max_{v\in V} \frac{d_v^2}{m_v} + m_v$
6. O $\max_{v\in V} \sqrt{m_v^2 + 3d_v^2}$
7. O $\max_{v\in V} \frac{d_v^2}{m_v} + d_v$
8. O $\max_{v\in V} \sqrt{d_v\left(m_v + 3d_v\right)}$
9. O $\max_{v\in V} \frac{m_v+3d_v}{2}$
10. O $\max_{v\in V} \sqrt{d_v\left(d_v + 3m_v\right.}$
11. O $\max_{v\in V} \frac{2m_v^3}{d_v^2}$
12. O $\max_{v\in V} \sqrt{2m_v^2 + 2d_v^2}$
13. O $\max_{v\in V} \frac{2m_v^4}{d_v^3}$
14. O $\max_{v\in V} \frac{2d_v^3}{m_v^2}$
15. X $\max_{v\in V} \sqrt{\frac{4m_v^3}{d_v}}$
16. O $\max_{v\in V} \frac{2d_v^4}{m_v^3}$
17. X $\max_{v\in V} \sqrt[4]{5d_v^4 + 11m_v^4}$
18. O $\max_{v\in V} \sqrt{\frac{2m_v^3}{d_v} + 2d_v^2}$
19. O $\max_{v\in V} \sqrt[4]{4d_v^4 + 12d_v m_v^3}$
20. O $\max_{v\in V} \frac{\sqrt{7d_v^2+9m_v^2}}{2}$
21. O $\max_{v\in V} \sqrt{\frac{d_v^3}{m_v} + 3m_v^2}$
22. O $\max_{v\in V} \sqrt[4]{2d_v^4 + 14d_v^2 m_v^2}$
23. O $\max_{v\in V} \sqrt{d_v^2 + 3d_v m_v}$
24. O $\max_{v\in V} \sqrt[4]{6d_v^4 + 10m_v^4}$
25. O $\max_{v\in V} \sqrt[4]{3d_v^4 + 13d_v^2 m_v^2}$
26. O $\max_{v\in V} \frac{\sqrt{5d_v^2+11d_v m_v}}{2}$
27. O $\max_{v\in V} \sqrt{\frac{3d_v^2+5d_v m_v}{2}}$
28. X $\max_{v\in V} \sqrt{\frac{2m_v^4}{d_v^2} + 2d_v m_v}$
29. X $\max_{v\in V} \sqrt{m_v^2 + \frac{3m_v^3}{d_v}}$
30. O $\max_{v\in V} \frac{m_v^3}{d_v^2} + \frac{d_v^2}{m_v}$
31. X $\max_{v\in V} \frac{4m_v^2}{m_v+d_v}$
32. X $\max_{v\in V} \frac{\sqrt{m_v^3(m_v+3d_v)}}{d_v}$
33. O $\max_{v_i\sim v_j} 2\left(d_i + d_j\right) - \left(m_i + m_j\right)$

34. O $\max_{v_i \sim v_j} \frac{2\left(d_i^2+d_j^2\right)}{d_i+d_j}$

35. O $\max_{v_i \sim v_j} \frac{2\left(d_i^2+d_j^2\right)}{m_i+m_j}$

36. X $\max_{v_i \sim v_j} \frac{2\left(m_i^2+m_j^2\right)}{d_i+d_j}$

37. O $\max_{v_i \sim v_j} \sqrt{2\left(d_i^2+d_j^2\right)}$

38. O $\max_{v_i \sim v_j} 2+\sqrt{2\left(d_i-1\right)^2+2\left(d_j-1\right)^2}$

39. O $\max_{v_i \sim v_j} 2+\sqrt{2\left(d_i^2+d_j^2\right)-4\left(m_i+m_j\right)+4}$

40. O $\max_{v_i \sim v_j} 2+\sqrt{2\left(\left(m_i-1\right)^2+\left(m_j-1\right)^2\right)+\left(d_i^2+d_j^2\right)-\left(d_i m_i+d_j m_j\right)}$

41. X $\max_{v_i \sim v_j} 2+\left(m_i+m_j\right)-\left(d_i+d_j\right)+\sqrt{2\left(d_i^2+d_j^2\right)-4\left(m_i+m_j\right)+4}$

42. O $\max_{v_i \sim v_j} \sqrt{d_i^2+d_j^2+2 m_i m_j}$

43. X $\max_{v_i \sim v_j} 2+\sqrt{3\left(m_i^2+m_j^2\right)-2 m_i m_j-4\left(d_i+d_j\right)+4}$

44. O $\max_{v_i \sim v_j} 2+\sqrt{2\left(\left(d_i-1\right)^2+\left(d_j-1\right)^2+m_i m_j-d_i d_j\right)}$

45. X $\max_{v_i \sim v_j} 2+\sqrt{\left(d_i-d_j\right)^2+2\left(d_i m_i+d_j m_j\right)-4\left(m_i+m_j\right)+4}$

46. O $\max_{v_i \sim v_j} 2+\sqrt{2\left(d_i^2+d_j^2\right)-16 \frac{d_i d_j}{m_i+m_j}+4}$

47. O $\max_{v_i \sim v_j} \frac{2\left(d_i^2+d_j^2\right)-\left(m_i-m_j\right)^2}{d_i+d_j}$

48. X $\max_{v_i \sim v_j} \frac{2\left(d_i^2+d_j^2\right)}{2+\sqrt{2\left(d_i^2+d_j^2\right)-4\left(m_i+m_j\right)+4}}$

49. X $\max_{v_i \sim v_j} 2+\sqrt{2\left(m_i^2+m_j^2\right)+\left(d_i-d_j\right)^2-4\left(d_i+d_j\right)+4}$

50. X $\max_{v_i \sim v_j} 2 \frac{d_i^2+d_j^2+m_i m_j-d_i d_j}{d_i+d_j}$

51. X $\max_{v_i \sim v_j} 2\left(m_i+m_j\right)-4 \frac{m_i m_j}{d_i+d_j}$

52. X $\max_{v_i \sim v_j} 2+\sqrt{\sqrt{8\left(m_i^4+m_j^4\right)-8\left(d_i^2+d_j^2\right)+4}-4\left(d_i+d_j\right)+6}$

53. X $\max_{v_i \sim v_j} 2+\sqrt{\sqrt{8\left(m_i^4+m_j^4\right)-8\left(d_i m_i+d_j m_j\right)+4}-4\left(d_i+d_j\right)+6}$

54. X $\max_{v_i \sim v_j} 2+\sqrt{2\left(m_i^2+m_j^2\right)+\left(d_i m_i+d_j m_j\right)-\left(d_i^2+d_j^2\right)-4\left(d_i+d_j\right)+4}$

55. X $\max_{v_i \sim v_j} 2+\sqrt{3\left(m_i^2+m_j^2\right)-\left(d_i^2+d_j^2\right)-4\left(m_i+m_j\right)+4}$

56. O $\max_{v_i \sim v_j} \frac{\left(d_i^2+d_j^2\right)\left(m_i+m_j\right)}{2 d_i d_j}$

57. X $\max_{v_i \sim v_j} 2+\sqrt{2\left(m_i^2+m_j^2\right)-8 \frac{d_i^2+d_j^2}{m_i+m_j}+4}$

58. X $\max_{v_i \sim v_j} 2+\sqrt{2\left(m_i^2+m_i m_j+m_j^2\right)-\left(d_i m_i+d_j m_j\right)-4\left(d_i+d_j\right)+4}$

59. X $\max_{v_i \sim v_j} \frac{2\left(m_i^2+m_i m_j+m_j^2\right)-\left(d_i^2+d_j^2\right)}{m_i+m_j}$

60. X $\max_{v_i \sim v_j} 2 + \sqrt{2\left(m_i^2 + m_i m_j + m_j^2\right) - \left(d_i^2 + d_j^2\right) - 4\left(d_i + d_j\right) + 4}$

61. X $\max_{v_i \sim v_j} \frac{2\left(m_i^2+m_j^2\right)}{2+\sqrt{2\left((d_i-1)^2+(d_j-1)^2\right)}}$

62. X $\max_{v_i \sim v_j} 2 + \sqrt{m_i^2 + 4m_i m_j + m_j^2 - 2d_i d_j - 4\left(d_i + d_j\right) + 4}$

63. X $\max_{v_i \sim v_j} d_i + d_j + m_i + m_j - 4\frac{d_i d_j}{m_i+m_j}$

64. X $\max_{v_i \sim v_j} \frac{m_i m_j (d_i+d_j)}{d_i d_j}$

65. X $\max_{v_i \sim v_j} \frac{(m_i+m_j)(d_i m_i+d_j m_j)}{2m_i m_j}$

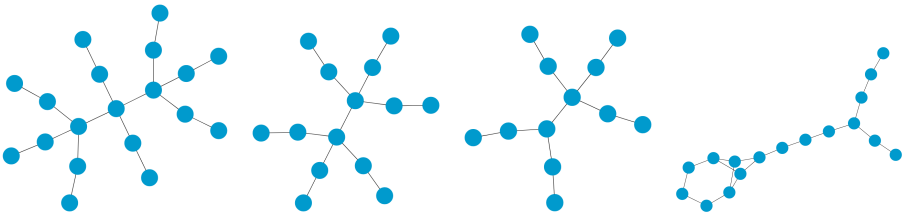
66. X $\max_{v_i \sim v_j} \frac{m_i^2+4m_i m_j+m_j^2-(d_i m_i+d_j m_j)}{d_i+d_j}$

67. X $\max_{v_i \sim v_j} \frac{(m_i+m_j)(d_i m_i+d_{jj} m_j)}{2d_i d_j}$

68. X $\max_{v_i \sim v_j} 2 + \sqrt{\left(m_i - m_j\right)^2 + 4d_i d_j - 4\left(m_i + m_j\right) + 4}$

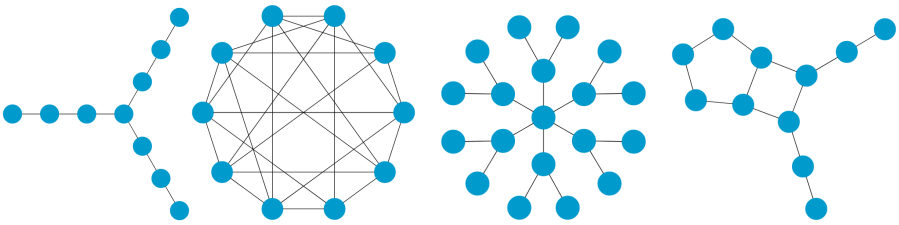
Appendix C. Counter-examples

Some counter-examples for each of the 29 refuted conjectures.



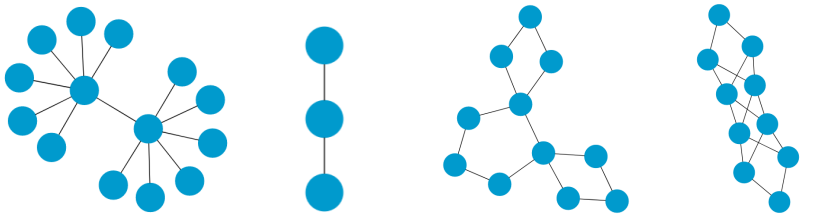
(a) Conjectures 3, 28 and 31 (b) Conjecture 15 (c) Conjecture 29 (d) Conjecture 36

Figure C1. Conjectures 3, 15, 28, 31, 29, and 36



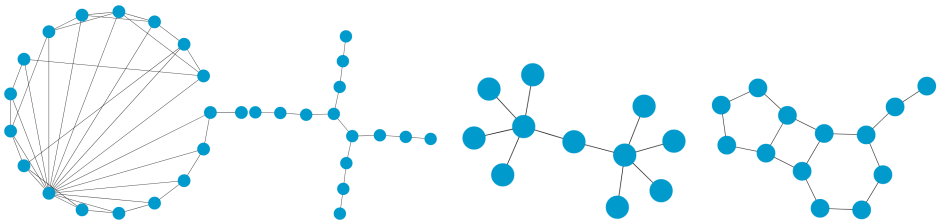
(a) Conjectures 36, 43, 49, 55, 58, 59, 60, 63, 66 and 67 (b) Conjecture 41 (c) Conjecture 41 (tree) (d) Conjecture 43

Figure C2. Conjectures 36, 41, 41 (tree), 43, 45 and others



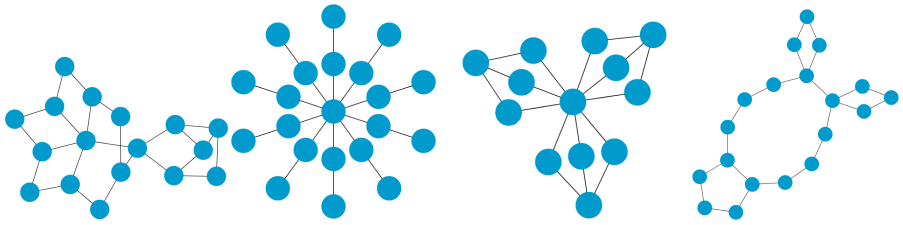
(a) Conjecture 45 (b) Conjectures 48, 57 and 61 (c) Conjectures 49 (d) Conjecture 50

Figure C3. Conjectures 45, 48, 57, 61, 49, and 50



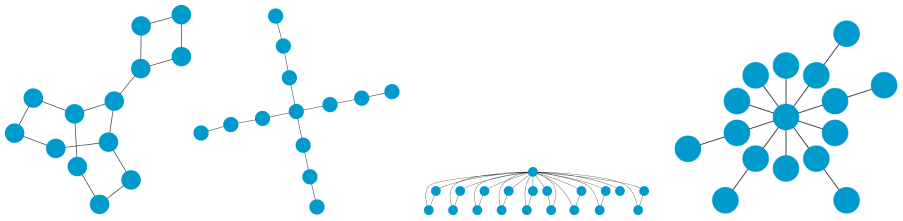
(a) Conjecture 51 (b) Conjectures 52, 53, 62 (c) Conjecture 54 (d) Conjectures 58

Figure C4. Conjectures 51, 52, 53, 62, 54, and 58



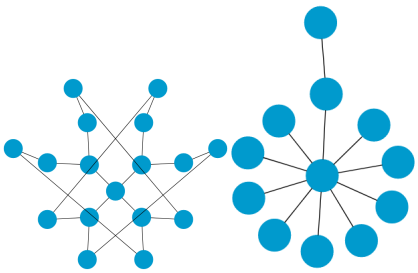
(a) Conjecture 58 (bigger) (b) Conjecture 60 (c) Conjectures 63 (d) Conjectures 63 (bigger)

Figure C5. Conjectures 58 (bigger), 60, 63 and 63 (bigger)



(a) Conjecture 64 (b) Conjectures 64 (tree) (c) Conjecture 65 (d) Conjecture 65 (tree)

Figure C6. Conjectures 64, 64 (tree), 65, and 65 (tree)



(a) Conjectures 66 (b) Conjectures 68

Figure C7. Conjectures 66 and 68

2.2 Transportation Network Optimization

The spectrum of graph-related matrices is not only used in conjectures but can also be used as a metric for robustness. I have a special interest in graph theory and building graphs. In the following paper, the construction of graphs was not targeted at refuting conjectures, but at generating better graphs for transportation and communication. This time the optimization process starts from a pre-existing (real or randomly generated) graph and adds edges in order to maximize the efficiency and robustness of the graph, but the process is limited by a budget: the sum of the lengths of the added edges must not exceed it. The conclusions of this article are multiple. Variants of the problem behave in very different manners, sometimes favoring greedy approaches, sometimes policy learning, and sometimes regular Monte Carlo Search. Robustness is best optimized using a greedy and deterministic beam search, while the novel variant of the Nested Monte Carlo Search, the Lazy Nested Monte Carlo Search, provides good performance for efficiency and robustness.

The state of the art is greatly outperformed, and the problems serve as an interesting benchmark to showcase each algorithm's capabilities due to the different ways it behaves under different parameters. In the end it cements the new LNMCS variant of NMCS as one of the best-performing MCTS algorithms available.

BUDGET LIMITED SPATIAL NETWORK IMPROVEMENT FOR TRANSPORT AND COMMUNICATIONS USING MONTE CARLO SEARCH

Abstract. We propose a comparison of a large selection of state-of-the-art deterministic and Monte Carlo Search (MCS) algorithms on the budget-limited network optimization problem. Including a new one producing better results, and different approaches to simplifying this problem. We show that not one algorithm can be the only answer for optimizing both efficiency and robustness, but that a simple heuristic can perform well for the optimization of robustness, and that the LNMCS with greedy playout is a reliable choice for optimizing efficiency on 100 nodes large graphs. We provide results for procedurally generated graphs and real-world ones, and show that results obtained on synthetic graphs match with to real-world graphs.

Key words. Monte Carlo Search, Graph, Network, Generation

MSC codes. 05-04, 05-08, 05B30, 05C12, 05C40, 68Q25, 68Q87, 68R05, 68R10, 68T05, 68T20, 68W20, 68W40

1. Introduction. Optimizing transportation networks is one of the most important real-world problems engineers have to face. Critical infrastructure, such as the internet or the roads, efficiencies and robustness rely on optimizing these types networks.

Search algorithms are already making most of the State of the Art on this problem. More precisely, stochastic search algorithms dominate the network optimization problem due to the large search space. We identified two families of stochastic search algorithms for network optimization: genetic algorithms [9, 15], and Monte Carlo Search algorithms [2].

Be it communications or transportation, the optimization must respect constraints such as the budget or the topology. In this paper, we compare the performances of multiple algorithms over synthetic and real world instances of the problem. Classic deterministic ones such as Beam search or Best First Search (BFS). Against Monte Carlo Search ones, namely Upper Confidence bounds applied to Trees (UCT) and others such as Nested Monte Carlo Search (NMCS), Nested Rollout Adaptation (NRPA), and Rapid Action Value Estimation (RAVE). As transportation and communication networks have to be efficient and resilient, we chose to optimize them over one metric focused on robustness, and another focused on efficiency.

2. The Network Design Problem. Our instance of the Network Design Problem (NDP) [18] consists of a pre-existing weighted graph with infinite capacity. The weight on the edges represents the cost of going to the node on one side of the edge from the node on the other side of the edge. Here the cost is the distance between the two nodes. It is the kind of graph we can use Dijkstra and Floyd-Warshall algorithms on. We do not look into the flow optimization in this paper, only the topological properties.

Our goal is, given such a graph, to optimize selected metrics on it. The optimization process starts from a graph and adds new edges until the budget is not exhausted, each edge added subtracts its length to the budget. The actions a search algorithm can select are the edges that are absent from the graph. The final states are states where the budget is too low to add any new edge. Here we chose to maximize specific definitions of the efficiency [8] and of the robustness [4].

Efficiency is defined as in Latora's work [8] : $E(G) = \frac{1}{N(N-1)} \sum_{i \neq j} sp(i,j)$ with

N the size of the graph, and $sp(i, j)$ the shortest path between vertices i and j. Just like in [2], we divide this value by the ideal efficiency, the efficiency of the complete graph, to obtain a value between 0 and 1.

However, we decided to use another metric for robustness than the one used in [2]. We think this metric for robustness is too computationally costly (we can not compare our results directly with theirs anyway, see section 4). We instead decided to use the much less costly spectral radius of the adjacency matrix λ_1. It is the largest eigenvalue and is described as a powerful robustness estimation in [4], in this survey's own words:

"The spectral radius is closely related to the path capacity or loop capacity of the graph. That is, the number of walks of length k (k = 2, 3, 4...) gives an indication of how well connected the graph is. If the graph has many loops and paths, then the graph is well connected i.e., larger λ_1."

"As a robustness measure, a larger λ_1 indicates a more robust graph to random failures and attack, along with increased susceptibility to virus propagation."

This is not the only relevant metric of robustness present in [4]. It is also stated that the average distance between stops (i.e. the efficiency) can be a good robustness metric, and that spectral-based techniques are scalable to larger graphs, which is one of our goals here.

2.1. Synthetic instances. To compare our algorithm we need instances of various sizes. We used the same method as used in [2] which itself was from [7]. This method is as follows:
(1) place a node u with a random positions in $[0, 1]^2$
(2) if there are other nodes, connect this node to each of them (v) given the probability $p(u, v) = \beta e^{-\alpha d(u,v)}$
(3) if u was not connected remove it
(4) if there's less than the desired amount of nodes, go to (1)

We use the same parameters as in [2]: $\alpha = 10$ and $\beta = 0.001$

The networks generated this way have the advantage of resembling real-world networks as you can see in figure 1. They however feature overlapping edges which are absent from real-world networks.

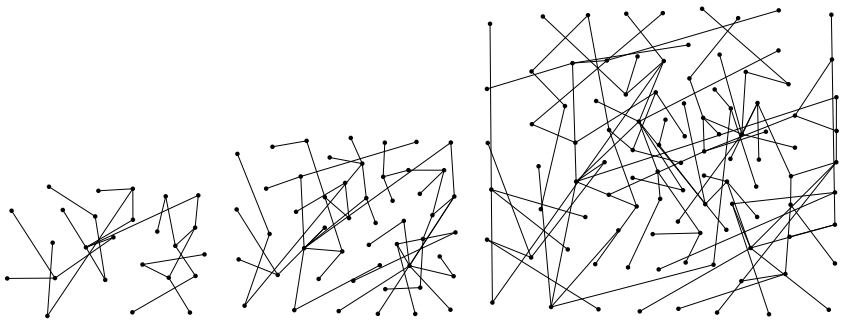


Fig. 1: Graphs of size 25, 50, 100 generated by seed 0

3. Algorithms.

3.1. Playout. Playouts or Rollouts, are the core elements of most Monte Carlo Search (MCS) algorithms. They are used to evaluate a state by simulating a the rest of the "game" from it.

The algorithm of an all-purpose playout is featured in algorithm 3.1.

Playouts are usually uniformly random and return a terminal state ($classicPlayout = true$ in algorithm 3.1). However, if the problem admits any state encountered during the simulation as a potential solution, the best state encountered can be returned ($classicPlayout = false$). The playout can also not be uniformly random, but can be guided by the immediate gain ($greedy = true$). Finally, the playouts can be guided by a learned policy like with the Nested Rollout Policy Adaptation (NRPA) algorithm.

Algorithm 3.1 Playout algorithm.

Function `Playout(`*state*`):`
  b-score $\leftarrow -\infty$ $\sigma^* \leftarrow ()$ *c-st* $\leftarrow$ *state* $\sigma \leftarrow ()$ **while** *c-st is not terminal* **do**
    if *greedy* **then** $move \leftarrow Greedy(M(c\text{-}st))$;
    else $move \leftarrow Random(M(c\text{-}st))$;
    c-st $\leftarrow$ $play(c\text{-}st, move)$ $\sigma.push(move)$ **if** *b-score* $\leq$ *c-st.score* *or* *classicPlayout* **then**
      b-score $\leftarrow$ *c-st.score* $\sigma^* \leftarrow \sigma$
    end
  end
  return *b-score*, σ^*

3.2. BEAM. Beam search is a simple baseline tree search algorithm. It only takes one integer parameter, the width w. It necessitates having access to a reliable evaluation of the search space states, otherwise the evaluation can be made with playouts. The algorithm keeps in memory w nodes, opens and evaluates all the children of these nodes, and then replaces the previous w nodes with the w best nodes recently opened. It has the advantage of forcing progress deep into the tree.

A beam search with $width = 1$ is what we call a greedy playout. A beam search of width w is usually w times more computationally heavy than a beam search of width 1, thus greedy playouts can be replaced by beam searches in many algorithms with a linear increase in computation costs.

3.3. GBFS. Greedy Best First Search is another simple yet effective baseline tree search algorithm [3]. It necessitates having access to a reliable way to evaluate nonterminal states, using playouts as a substitute if it is not possible, like Beam search. The algorithm selects the node with the best evaluation from a queue, evaluates all its children, and inserts them into the queue according to their evaluations. Thus, GBFS is guaranteed to explore only once all the nodes in the search tree, unlike BEAM search which can miss states with great valuations. Missing states with great valuations is usually unavoidable and not a shortcoming since the problems we are facing are generally NP-Hard and could never be solved exhaustively anyways.

3.4. UCT. The Upper Confidence bound applied to Trees is the most widely used instance of Monte Carlo Tree Search, and the default algorithm one goes to when using Monte Carlo Methods. It, or variants of it, is used in groundbreaking applications such as Deepmind's Alphago [16] or AstraZeneca's Aizynthfinder [6], and other [17].

Contrary to BEAM and GBFS, MCTS do not need a way to evaluate any state

from the search tree, and only need to evaluate final states. This is especially useful in games, like chess, where a function evaluating a game in progress is not trivial. But even when a such function is available, MCTS algorithms are generally better performing because they are more capable of avoiding local maximum and other traps that come with a noisy search space.

UCT and all the other MCTS algorithms share the same 4 phases:

(1) selection: until an unknown node is opened, go down the search tree according to the exploration/exploitation formula. (2) expansion: add a new node to the search tree (3) simulation: evaluate the newly added node, using playouts to get a terminal state (4) backpropagation: use the simulation result to update the values of all the search tree nodes visited during the selection phase

In UCT's case, the exploration/exploitation formula for each move m from a state s is: $\frac{w}{n} + c * \sqrt{\frac{lnN}{n}}$

Where w is the sum of all scores obtained after playing m from s, n is the number of times m was played from s, and N is the number of times s was visited during the selection phase. c is a constant (usually $c \approx 1$).

3.5. RAVE and GRAVE. Rapid Action Value Estimation (RAVE) is an MCTS algorithm derived from UCT and introduced by Sylvain Gelly and David Silver [5]. The main difference with UCT is that RAVE generalizes the value of moves over the entire search tree (for example, if a move generally leads to better results, then it may be favored even if in the UCT-like subtree it led to worse results). This is adapted to problems where the order of the moves is less important, for example, go and not chess. We think the network optimization problem is appropriate.

The Generalized Rapid Action Value Estimation (GRAVE) is a generalization of RAVE. Unlike RAVE, when deciding which move to play, it inherits a policy from the last parent move whose children experienced more than ref playouts for more localized generalization. It can also be used in conjunction with a move selection heuristic.

3.6. NMCS. Nested Monte Carlo Search is another type of Monte Carlo Search algorithms, different from MCTS like UCT, PUCT, RAVE, and GRAVE in their iterative nature, NMCS is recursive.

The NMCS calls lower-level NMCS on all of the currently available moves from the current state. Each NMCS returns the best path it found to optimize the value of the state. The higher-level NMCS then executes the first action from the best path it has in memory and calls new lower-level NMCS on the available moves from the resulting state.

Compared to UCT, NMCS has the advantage of optimizing at any depth of the search tree and not only near the root. It generally shows better results on optimization problems [12] [13].

3.7. LNMCS. The Lazy Nested Monte Carlo Search presented in [12] is a variant of NMCS made to address one of its shortcomings. A NMCS of level l requires computing as many NMCS of level $l-1$ as the number of moves available from the state, and then repeating that for each level of depth of the search tree. The computation time of the NMCS increases greatly with the level, an NMCS of level over 3 or even 2 can be too computationally costly depending on the problem.

Under the assumption that some moves doom the lower-level NMCS to underwhelming results, we decide to reintroduce the exploration-exploitation dilemma present in MCTS to the NMCS, and prune some of the lower-level NMCS based on cheap

and relative evaluations using playouts.

Before calling a lower-level LNMCS, the available moves are each sampled with b playouts. If the mean of the evaluations of a move is inferior to the mean of all the evaluations made on that depth tr plus the rate r times the difference between the best evaluation ever encountered on that depth $trmax$ and the mean of all the evaluations made on that depth tr, then it is pruned: a single playout is launched instead of a lower-level LNMCS. For example on Figure 2, the middle and right moves are sampled with good enough results to pass the threshold and their lower-level LNMCS are called, while the leftmost move has poor sampling and is pruned.

This means that a rate r of 0 prunes all the moves inferior to the mean of the evaluations on a certain depth.

You can find a pseudocode for LNMCS in [12].

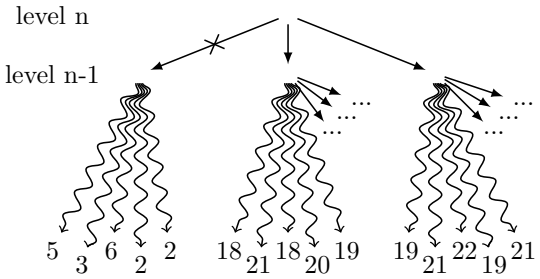


Fig. 2: Level n LNMCS pruning a search subtree and launching n-1 LNMCS on surviving search subtrees.

3.8. NRPA. The Nested Rollout Policy Adaptation is a MCTS algorithm introduced by Cristopher Rosin [11] in 2011. It is derived, but very different, from the NMCS. It uses the nesting not to progress deeper into the tree, but to contribute hierarchically to a policy that is learned from playouts (or rollouts) to guide future playouts.

4. Results.

4.1. Preliminary results.

4.1.1. Variation among the synthetic instances. Before diving into the performances of our algorithms on the synthetic benchmark instances, we think it's crucial to sample these possible synthetic instances to know more about their depth and potential scores. To do so, we generated 20 synthetic instances of size 25 with $\alpha = 10$ and $\beta = 0.001$ and maximized their efficiency using an exhaustive algorithm with a budget of 0.1 times the total cost of the starting edges as in [2].

With as little as 15206 and as much as 18222026 explored states, even small 25 nodes graphs can take hours to be explored exhaustively, or a few minutes. The search space size can vary greatly with a standard deviation of 4183678.3, mainly due to extreme outliers.

The starting efficiency and their improvement show a significant standard deviation too.

With such variations among all the parameters surrounding the synthetic instances, it appears necessary to compare our algorithms on multiple instances and

seed	search tree size	start efficiency	efficiency gain
0	180557	0.575	0.222
1	453598	0.498	0.304
2	1118035	0.511	0.279
3	19353	0.600	0.190
4	3990749	0.423	0.354
5	16593	0.585	0.203
6	196488	0.289	0.523
7	451586	0.427	0.301
8	6662608	0.431	0.349
9	15206	0.158	0.571
10	849969	0.604	0.226
11	18222026	0.305	0.498
12	202226	0.468	0.295
13	744556	0.475	0.305
14	144717	0.491	0.312
15	1538641	0.432	0.322
16	247973	0.527	0.273
17	3326338	0.540	0.226
18	443276	0.455	0.340
19	334322	0.529	0.331
Mean	1.957e6	0.466	0.321
Std dev	4.183e6	0.112	0.103

Table 1: Differences in search tree sizes, starting and best values among 20 synthetic graphs of size 25 with $\alpha = 10$ and $\beta = 0.001$

share them to help with reproducibility and improve the relevancy of said results. There seems to be no correlation between the size of the search tree and the efficiency gain over this small sample, however, the start efficiency and the efficiency gain seem to sum around 0.8 with these parameters.

Given [2] results show almost no variation, we assume they realized all their experiments on a graph generated from a unique seed.

4.1.2. Hyperparameter tuning and algorithmic choice. Before diving into lengthy experiments, we first need to quickly evaluate the algorithms likely to perform well and how much the results would vary

We launched UCT with $c = 1.0$ and NMCS with $level = 2, 3$, 10 times over the graph of size 50 generated by the seed 0 with the previously mentioned parameters and budget with a timeout of 600 seconds, the initial score is 0.43899. Over our 10 experiments, we obtain final efficiencies featured in tables 2. This subsection's experiments were realized on an Intel i5-6600K 3.50GHz CPU, which is different from the CPU used in section 4.2.

It is interesting to note that UCT peaks around the 120th second, and NMCS continues to find better results after the 400th second.

We do not include the standard deviation in subsequent results as it is always between 0.005 and 0.015 and impairs readability. We think it is low enough to justify using the means over 10 runs.

experiment	UCT	NMCS 2	NMCS 3
1	0.56436	0.57813	0.59877
2	0.55633	0.58891	0.59768
3	0.56271	0.59257	0.58098
4	0.55491	0.57611	0.58943
5	0.55557	0.58085	0.57400
6	0.56734	0.58652	0.58554
7	0.55697	0.57866	0.57649
8	0.57557	0.60954	0.58618
9	0.55668	0.60097	0.59544
10	0.56689	0.60194	0.57162
Mean	0.56173	0.58942	0.58561
Std dev	0.00682	0.01158	0.00980

Table 2: Preliminary comparison between NMCS and UCT and evaluation of the variance on one seed

4.2. On the synthetic instances.

4.2.1. Experimental setup. To evaluate each algorithm, we launch each of them 10 times over 3 different sizes: 25, 50, and 100. We then repeat this experiment 5 times, for each graph generating seed from 0 to 4. In tables 3, 4, and 5 we display the results for directly solving the efficiency problem with each algorithm each size.

In tables 6, 8, and 7 we try new approaches to try to maximize the final efficiency.

In tables 12, 14, and 13 we apply the previously best methods on the robustness problem and compare it against few representative baseline methods.

As seen in figure 1, these networks seem to imperfectly model networks like streets as edges can overlap.

These experiments were made with Rust 1.59, on an Intel Core i7-11850H 2.50GHz using a single core.

We compare the following algorithms:

- UCT, with $c = 1$ as the baseline MCTS algorithm
- GBFS, a baseline greedy deterministic algorithm
- BEAM, with widths of 10, 50, and 100, another baseline greedy deterministic algorithm
- NMCS, with $l = 2$ and another with $l = 3$ given its good performances and relative simplicity
- LNMCS, with the default hyperparameters $l = 3$, $r = 0.8$ and $p = 3$ as it usually improves over NMCS [14]
- NRPA, with $l = 3$
- RAVE, as a recent improvement over UCT
- GRAVE, $ref = 50$ is shown to be a good value in [1]
- GRAVE B with $ref = 50$, $bias = 10$ and using the cost-effectiveness ($value_{change}/cost$) as a move selection heuristic.

4.2.2. Solving the problem directly. As you can see in Table 1, a network size of 25 is small enough for the BFS to find the optimal value with all seeds except seed 2, which features a large search tree. The optimal value is found on seed 4 despite an even larger search tree. Beam search features good results but they are not

seed	0	1	2	3	4
start value	0.575	0.498	0.511	0.600	0.423
BFS	**0.797**	**0.802**	**0.791**	**0.790**	**0.778**
BEAM 10	0.768	0.757	0.736	0.778	0.749
BEAM 50	0.781	**0.802**	0.747	**0.790**	0.770
BEAM 100	0.766	0.769	0.786	**0.790**	0.770
LNMCS	0.755	0.754	0.729	0.764	0.763
NMCS 2	0.769	0.752	0.757	0.772	0.752
NMCS 3	0.791	0.786	0.781	0.789	0.764
NRPA	0.792	0.776	0.770	**0.790**	0.719
UCT	0.724	0.748	0.677	0.746	0.770
RAVE	0.711	0.612	0.687	0.716	0.623
GRAVE	0.704	0.624	0.680	0.702	0.641
GRAVE B	0.708	0.632	0.693	0.709	0.611

Table 3: Final efficiencies found for each algorithm on each graph of size 25 after 600s

seed	0	1	2	3	4
start value	0.438	0.398	0.439	0.445	0.373
BFS	0.607	0.619	0.623	0.639	0.587
BEAM 10	0.614	0.543	0.583	0.668	0.587
BEAM 50	**0.659**	0.624	**0.659**	0.687	0.604
BEAM 100	0.655	0.620	0.642	**0.700**	**0.627**
LNMCS	0.647	**0.632**	0.637	0.681	0.621
NMCS 2	0.628	0.583	0.637	0.673	0.561
NMCS 3	0.641	0.589	0.646	0.692	0.588
NRPA	0.644	0.577	0.650	0.677	0.570
UCT	0.590	0.542	0.602	0.580	0.541
RAVE	0.545	0.510	0.547	0.592	0.487
GRAVE	0.556	0.505	0.570	0.589	0.518
GRAVE B	0.558	0.506	0.557	0.589	0.513

Table 4: Final efficiencies found for each algorithm on each graph of size 50 after 600s

optimal, the problem cannot always be solved optimally by greedy playouts. Among the Monte Carlo algorithms, NMCS 3 and NRPA show the best results, the UCT family is lagging behind the nested family. Over 10 runs, the NRPA managed to always find the optimal solution on seed 3, which is impressive for a Monte Carlo algorithm.

With a network size of 50, the BFS is no longer able to explore most of the tree, and larger widths are required for the beam search to produce good results. LNMCS becomes the best algorithm among the MCTS (except on seed 2). When the search tree size increases, pruning bad subtrees becomes more efficient.

As the search tree size increases further with networks of size 100, on table 5, LNMCS becomes the best algorithm and is only outperformed by BFS on seed 3 and BEAM 10 on seed 2. BEAM 10 being a very close second is interesting because it is both a simple and a greedy algorithm. This result is what pushes us to investigate

seed	0	1	2	3	4
start value	0.307	0.343	0.357	0.382	0.329
BFS	0.456	0.464	0.477	**0.511**	0.444
BEAM 10	**0.459**	0.476	**0.517**	0.484	0.429
BEAM 50	0.440	0.436	0.449	0.467	0.411
BEAM 100	0.416	0.421	0.439	0.458	0.392
LNMCS	**0.459**	**0.480**	0.497	0.505	**0.461**
NMCS 2	0.443	0.451	0.476	0.479	0.425
NMCS 3	0.437	0.453	0.473	0.479	0.431
NRPA	0.446	0.451	0.471	0.484	0.426
UCT	0.406	0.424	0.441	0.449	0.399
RAVE	0.405	0.412	0.441	0.449	0.386
GRAVE	0.406	0.420	0.437	0.469	0.409
GRAVE B	0.373	0.425	0.422	0.456	0.398

Table 5: Final efficiencies found for each algorithm on each graph of size 100 after 600s

the use of greedy playouts instead of random playouts.

4.2.3. Greedy playouts and action space reduction. One of the main obstacles we encountered with these experiments is the width of the search tree: with a size of 50, the number of available moves is around a thousand at each state. This poses a problem to the NMCS (which achieves better performance than UCT despite that) as it means billions of score computations. Even LNMCS encounters difficulties in properly evaluating each of these moves when its pruning capabilities help to alleviate this problem.

Inspired by PUCT, which uses a prior neural network to suggest a smaller set of moves when the number of playable moves is too large (like for go with Deepmind's alphago), we make our algorithms only consider the N cheapest moves (here N = 20). Many more expensive moves will thus not be used, it leads to better results as shown in [2].

Inspired by the beam search good performances in tables 4 and 5, even on larger networks, we aim to try greedy playouts on this problem.

We compare the best algorithms with greedy playouts, with action space reduction, and with greedy playouts and action space reduction combined.

The BFS, when using greedy playouts (GPBFS), is slightly modified to swap the node evaluation function to a single greedy playout.

NRPA is by definition applying a learned policy on the playout, replacing it with a greedy playouts would turn it into a simple sampling algorithm. The beam search does not use playouts at all. This is why NRPA and beam search are not featured in our experiments involving greedy playouts.

seed	0	1	2	3	4
start value	0.307	0.343	0.357	0.382	0.329
UCT	0.386	0.420	0.432	0.487	0.387
CSGUCT	0.386	0.420	0.432	0.487	0.387
GPBFS	**0.430**	0.449	**0.463**	**0.495**	0.405
LNMCS	0.399	0.449	**0.463**	**0.495**	0.405
NMCS 3	0.405	**0.453**	**0.463**	0.491	**0.407**

Table 6: Final efficiencies found with few algorithms on each graph of size 100 after 600s with greedy playouts

As you can see in table 6, using greedy playouts alone did not lead to better results. With large graphs and no action space reduction, it is required to compute the efficiency thousands of times per greedy playout, each one requiring applying the Floyd-Warshall algorithm, making each of the playouts very costly. Only a few greedy playouts can be played in 10 minutes (∼15s per greedy playout), which explains the redundancy of the results, LNMCS has not enough time to prune anything. Both UCT and CSGUCT produced the exact same results because they are very similar, GPBFS is slightly better performing but is worse than without greedy playouts too. We did not conduct any more experiments on greedy playouts alone because we speculate that the results will be inferior for all algorithms compared to their random playouts results.

seed	0	1	2	3	4
start value	0.307	0.343	0.357	0.382	0.329
BEAM 10	0.509	0.540	0.564	0.547	0.518
BFS	0.449	0.442	0.462	0.507	0.429
NMCS 3	0.528	0.548	0.555	0.588	**0.529**
LNMCS	0.499	0.545	0.560	0.593	0.522
NRPA	**0.538**	**0.560**	**0.565**	**0.597**	0.514
UCT	0.467	0.497	0.494	0.522	0.465

Table 7: Final efficiencies found with selected algorithms on each graph of size 100 after 600s with action space reduction

In table 7, all algorithms except BFS better results than in table 5. In addition, it is noticeable that the smaller number of available moves seems to help the NRPA achieve even better results. We suppose it is because the action space becomes small enough for the NRPA to learn a policy. Having a thousand actions available makes it harder to build up the policy. The gap between LNMCS and NMCS 3 results has disappeared compared to the results featured in table 5. This is unexpected since LNMCS usually performs better than NMCS on most problems. We suppose it might be due to the pruning of the expensive moves: with this setting all moves lead to good subtrees and LNMCS loses its advantage.

seed	0	1	2	3	4
start value	0.307	0.343	0.357	0.382	0.329
GPBFS	0.512	0.568	**0.597**	0.569	0.501
NMCS 3	**0.545**	0.493	0.569	0.537	0.458
LNMCS	0.519	**0.569**	0.592	**0.599**	**0.503**
UCT	0.483	0.502	0.489	0.541	0.470
CSGUCT	0.460	0.511	0.482	0.510	0.470

Table 8: Final efficiencies found with selected algorithms on each graph of size 100 after 600s with action space reduction and greedy playouts

In green, you can see the best results at maximizing the efficiency over all approaches for the graphs of side 100.

Combining both greedy playouts and reduction is featured in table 8. The results are almost always better than the direct approach, even the previously worse-performing algorithms can outperform the direct approach LNMCS (best algorithm on table 5). It also slightly outperforms the NRPA in the reduction only approach except on seed 4. The best algorithm using both action space reduction and greedy playouts is LNMCS, only outperformed by NMCS 3 on seed 0, and by GPBFS on seed 2. GPBFS is a very close second.

GPBFS and LNMCS using greedy playouts and action space reduction, and NRPA using action space reduction, are the three globally dominating approaches to the efficiency problem.

4.2.4. Robustness. Previously we showed that combining both greedy playouts and action space reduction led to generally better results. This is true for efficiency, in this sub-subsection we explore a different metric: robustness. Our goal is to determine if the best algorithms are roughly the same with this metric, and how a smaller computational cost of the metric affects the results.

Note that the efficiency used previously is the inverse of the average distance divided by the average distance of the complete version of the graph, according to [4] it is also linked to robustness. Here we need a robustness evaluation of lesser cost, we chose to use the spectral radius: the largest eigenvalue from the adjacency matrix.

seed	0	1	2	3	4
start value	2.512	2.861	3.153	2.738	3.146
BFS	**3.969**	**4.147**	**4.308**	**4.083**	**4.914**
BEAM 10	3.755	3.832	3.943	3.842	4.499
BEAM 100	**3.969**	**4.147**	4.145	**4.083**	4.689
LNMCS	3.728	3.898	3.832	3.932	4.407
NMCS 3	3.857	4.071	4.153	4.045	4.762
NRPA	3.759	3.876	4.045	4.025	4.487
UCT	3.755	3.987	3.488	3.778	3.968

Table 9: Final robustness found for each algorithm on each graph of size 25 after 600s

seed	0	1	2	3	4
start value	3.483	2.755	2.993	3.273	2.706
BFS	4.412	4.895	**4.809**	4.620	4.626
BEAM 10	4.286	4.734	4.383	4.667	5.354
BEAM 100	**5.313**	**5.000**	4.673	**5.097**	**5.782**
LNMCS	4.167	4.869	4.485	4.653	5.116
NMCS 3	4.593	4.880	4.663	4.726	5.114
NRPA	4.049	4.218	4.245	4.228	4.420
UCT	3.989	4.133	4.137	4.215	4.252

Table 10: Final robustness found for each algorithm on each graph of size 50 after 600s

seed	0	1	2	3	4
start value	2.964	3.397	3.085	3.265	3.500
BFS	4.215	5.242	5.071	4.551	5.537
BEAM 10	4.838	**5.467**	**5.144**	**5.627**	**5.633**
BEAM 100	3.788	4.090	4.019	4.164	4.241
LNMCS	**4.988**	5.353	5.134	5.328	5.444
NMCS 3	4.530	5.260	5.110	4.932	5.189
NRPA	3.961	4.222	4.032	4.156	4.303
UCT	3.780	4.111	3.944	4.029	4.145

Table 11: Final robustness found for each algorithm on each graph of size 100 after 600s

seed	0	1	2	3	4
start value	2.964	3.397	3.085	3.265	3.500
UCT	4.508	4.718	4.191	4.358	4.527
CSGUCT	4.508	4.718	4.191	4.358	4.527
NMCS 3	**4.878**	**5.355**	**4.614**	**4.803**	**5.007**
LNMCS	4.775	5.121	4.447	4.764	4.775
GPBFS	4.775	5.121	4.447	4.764	4.775

Table 12: Final robustness found with selected algorithms on each graph of size 100 after 600s with greedy playouts

seed	0	1	2	3	4
start value	2.964	3.397	3.085	3.265	3.500
BEAM 10	4.382	4.783	4.924	4.633	4.592
BFS	4.244	4.514	4.727	4.331	4.757
NMCS 3	**4.633**	**5.214**	**4.981**	4.654	4.674
LNMCS	4.207	4.868	4.974	4.514	4.664
NRPA	4.414	4.724	4.767	**4.825**	**4.856**
UCT	3.918	4.110	4.282	4.057	4.347

Table 13: Final robustness found with selected algorithms on each graph of size 100 after 600s with action space reduction

seed	0	1	2	3	4
start value	2.964	3.397	3.085	3.265	3.500
GPBFS	**4.802**	4.694	5.131	4.546	5.023
NMCS 3	4.743	5.233	4.672	4.593	4.426
LNMCS	4.772	**5.296**	**5.161**	**4.782**	**5.127**
UCT	4.134	4.319	4.477	4.280	4.387
CSGUCT	4.457	4.451	4.684	4.281	4.427

Table 14: Final robustness found with selected algorithms on each graph of size 100 after 600s with action space reduction and greedy playouts

In green, you can see the best results at maximizing the robustness over all approaches for the graphs of side 100.

Contrary to the good performances of a beam search of width 10 on table 11, using greedy playouts does not allow to beat these results on table 12. The action space is still too large for greedy layouts, only about 20 of them can be played without search space reduction in 10 minutes (30s per playout), which explains the good performances of NMCS over the other algorithms, the playouts are more spread.

The action space reduction does not lead to strictly better results (table 13) than the approach with greedy playouts only, the results are worse than the base approach too. Combined, the two approaches produce slightly better results (table 14) than isolated, but still inferior to the base approach.

Overall, to optimize this definition of the robustness over such graphs, the optimal way seems to use a beam search on the base approach (random playouts, full action space), a good second choice could be the LNMCS on the base approach too.

The goal of this section was not to determine which algorithms provide the best results for this specific robustness metric but to know if we may generalize the results of the various approaches tried on the efficiency over other metrics. We think these results are satisfactory because by trying only one other metric, we can say that the metrics are differently affected by the approaches. We know that greedy playouts and action space reduction are not universally better for this problem depending on the metric we want to optimize.

4.3. Real World Graphs. To better measure our algorithms on the network optimization problem, we decided to apply it to graphs from the Survivable Network Design Library, SNDlib [10]. This library features 26 networks, some represent cities

or countries, some are inspired by biology, and some are more abstract. We decided to run the best algorithms from the previous results on 4 graphs from the real world:

1. France, representing the country of France with 25 vertexes and 45 edges.
2. Germany50, representing the country of Germany with 50 vertexes and 88 edges.
3. India35, representing the country of India with 35 vertexes and 80 edges.
4. Cost266, representing the European Union with 37 vertexes and 57 edges.

For comparison, Germany50 is the third largest graph in the database, with only the abstract graphs of "brain" and "ta2" bigger with respectively 167 and 65 vertexes. In Figure 3 you can see what the graphs look like.

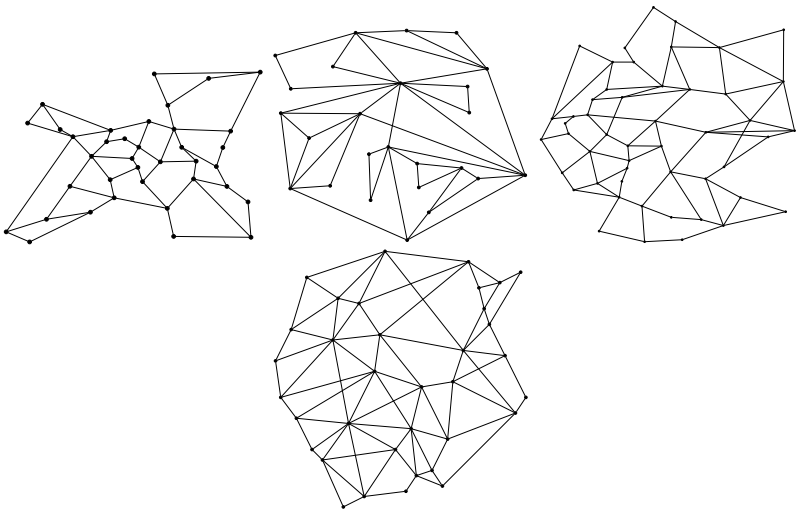


Fig. 3: Graphs from SNDlib for Europe, France, India and Germany from top-right clockwise.

Based on the previous results, we selected the best algorithms from each size for robustness and efficiency in table 15. LNMCS GR is the variant of LNMCS with greedy playouts and action space reductions as it is the one that performs the best overall for optimizing efficiency on large graphs, see table 8. As a baseline, we add the NMCS of level 3 with normal playouts and no action space reduction as it performed well enough in all the experiments.

	25	50	100
Robustness	BFS	BEAM 100	BEAM 10
Efficiency	BFS	BEAM 100	LNMCS GR

Table 15: Best algorithm for each size of the problem

graph	france	germany50	cost266	india35
start value	0.716590	0.848845	0.844898	0.953861
BFS	**0.884692**	0.883322	**0.912834**	**0.965096**
BEAM 100	0.876070	0.887081	**0.912834**	0.965004
LNMCS GR	0.872731	**0.895397**	0.909709	0.964143
NMCS 3	0.871349	0.885278	0.909772	0.964209

Table 16: Final efficiencies found with selected algorithms on each graph from SNDlib after 600s

graph	france	germany50	cost266	india35
start value	4.712265	4.085959	3.399925	5.491604
BFS	**5.594335**	5.102157	4.760420	**6.354730**
BEAM 100	5.505853	**5.325418**	**4.834799**	6.307945
BEAM 10	5.407345	5.218283	4.494036	6.271883
NMCS 3	5.565474	5.113068	4.799523	6.336223

Table 17: Final robustness found with selected algorithms on each graph from SNDlib after 600s

Except for NMCS 3, the results presented in Tables 16 and 17 are all from deterministic tree search algorithms so they were only executed once.

The results obtained in tables 17 and 16 corroborate the ones found on synthetic instances: BFS and BEAM give the best results for smaller graphs. Germany50 is the biggest graph and LNMCS performs already well for synthetic instances of size 50, this explains why this graph was best optimized with LNMCS with greedy playouts and action space reduction. NMCS 3 is a control experiment and was not expected to outperform the other algorithms. The graphs from SNDlib have 50 vertexes or less and do not give indications of whether the results obtained on synthetic instances apply similarly to graphs with 100 vertexes.

5. Conclusion. In this paper, we experimented with optimizing graphs for communications and transport under budget constraints. Multiple different graphs of different sizes were experimented upon, using two definitions of robustness and efficiency among many and with greedy playouts and action space reduction. We showed that while no algorithm is better than the others in any context, the LNMCS usually offers good enough results in most contexts. Deterministic greedy algorithms should not be ignored as the BEAM search offers great results even on larger graphs. Finally, synthetic graphs seem to behave similarly to real-world graphs, for sizes smaller than 50 vertexes at least.

REFERENCES

[1] T. Cazenave, *Generalized rapid action value estimation*, in 24th International conference on artificial intelligence, 2015, pp. 754–760.

[2] V.-A. Darvariu, S. Hailes, and M. Musolesi, *Planning spatial networks with monte carlo tree search*, Proceedings of the Royal Society A, 479 (2023), p. 20220383.

[3] J. E. Doran and D. Michie, *Experiments with the graph traverser program*, Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences, 294 (1966), pp. 235–259.

[4] S. Freitas, D. Yang, S. Kumar, H. Tong, and D. H. Chau, *Graph vulnerability and robustness: A survey*, IEEE Transactions on Knowledge and Data Engineering, (2022).

[5] S. Gelly and D. Silver, *Monte-carlo tree search and rapid action value estimation in computer go*, Artificial Intelligence, 175 (2011), pp. 1856–1875.

[6] S. Genheden, A. Thakkar, V. Chadimová, J.-L. Reymond, O. Engkvist, and E. Bjerrum, *AiZynthFinder: a fast, robust and flexible open-source software for retrosynthetic planning*, Journal of Cheminformatics, 12 (2020), p. 70, https://doi.org/10.1186/s13321-020-00472-1, https://jcheminf.biomedcentral.com/articles/10.1186/s13321-020-00472-1 (accessed 2022-08-11).

[7] M. Kaiser and C. C. Hilgetag, *Spatial growth of real-world networks*, Physical Review E, 69 (2004), p. 036103.

[8] V. Latora and M. Marchiori, *Efficient behavior of small-world networks*, Physical review letters, 87 (2001), p. 198701.

[9] C. Madapatha, B. Makki, A. Muhammad, E. Dahlman, M.-S. Alouini, and T. Svensson, *On topology optimization and routing in integrated access and backhaul networks: A genetic algorithm-based approach*, IEEE Open Journal of the Communications Society, 2 (2021), pp. 2273–2291, https://doi.org/10.1109/OJCOMS.2021.3114669.

[10] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski, *Sndlib 1.0—survivable network design library*, Networks: An International Journal, 55 (2010), pp. 276–286.

[11] C. D. Rosin, *Nested rollout policy adaptation for monte carlo tree search*, in In IJCAI, 2011, pp. 649–654.

[12] M. Roucairol., J. Arjonilla., A. Saffidine., and T. Cazenave., *Lazy nested monte carlo search for coalition structure generation*, in Proceedings of the 16th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART, INSTICC, SciTePress, 2024, pp. 58–67, https://doi.org/10.5220/0012302300003636.

[13] M. Roucairol and T. Cazenave, *Refutation of spectral graph theory conjectures with monte carlo search*, in Computing and Combinatorics, Y. Zhang, D. Miao, and R. Möhring, eds., Cham, 2022, Springer International Publishing, pp. 162–176.

[14] M. Roucairol and T. Cazenave, *Solving the hydrophobic-polar model with nested monte carlo search*, in International Conference on Computational Collective Intelligence, Springer, 2023, pp. 619–631.

[15] N. Shanmugasundaram, K. Sushita, S. P. Kumar, and E. Ganesh, *Genetic algorithm-based road network design for optimising the vehicle travel distance*, International Journal of Vehicle Information and Communication Systems, 4 (2019), pp. 344–354, https://doi.org/10.1504/IJVICS.2019.103931, https://www.inderscienceonline.com/doi/abs/10.1504/IJVICS.2019.103931, https://arxiv.org/abs/https://www.inderscienceonline.com/doi/pdf/10.1504/IJVICS.2019.103931.

[16] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. v. d. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, *Mastering the game of Go with deep neural networks and tree search*, Nature, 529 (2016), pp. 484–489.

[17] M. Świechowski, K. Godlewski, B. Sawicki, and J. Mańdziuk, *Monte carlo tree search: A review of recent modifications and applications*, Artificial Intelligence Review, 56 (2023), pp. 2497–2562.

[18] R. Wong, *A survey of network design problems*, (2004).

2.3 Coalition Structure Generation

The Coalition Structure Generation (CSG) is another combinatorial problem. This time, it is not about graphs but about forming coalitions of agents in order to maximize the sum of all coalition's values. Just like the network optimization problem, this problem has many parameters. Arguably too many, due to an over-abundance of coalition valuation functions brought by a single researcher. Despite this drawback, the CSG problem can be a good showcase for search algorithms under certain circumstances. While many valuation functions are vulnerable to greedy approaches, the original version of the problem, introduced by Tuomas Sandholm, a figure in operation research, AI, and combinatorics, is not vulnerable to greedy approaches.

No matter the valuation function, the Lazy Nested Monte Carlo Search is able to greatly outperform all state-of-the-art approaches. It is able to push the limits of this problem even further with a new representation of the search space.

In addition to showing the superiority of the LNMCS approach, this paper shows the unnecessity of several valuation functions, that were introduced for no reason, and clog the experimentations on this problem. Our results will likely be hard to outperform and we do not recommend working on this problem. All the other problems presented in this thesis are, in our opinion, more interesting to work on and more likely to yield interesting results.

Lazy Nested Monte Carlo Search for Coalition Structure Generation

Milo Roucairol*[1][a], Jérôme Arjonilla*[1][b], Abdallah Saffidine[2][c], and Tristan Cazenave [1][d]
* *These authors contributed equally to this work.*
[1] *Paris Dauphine University - PSL, Paris, France*
[2] *University of New South Wales, Sydney, Australia*
milo.roucairol@dauphine.eu, {jerome.arjonilla, tristan.cazenave}@dauphine.psl.eu, abdallah.saffidine@gmail.com

Keywords: Monte Carlo Search, Coalition Structure Generation, Algorithm, Agents, Combinatorial, Optimization

Abstract: This paper explores Monte-Carlo Search algorithms applied to Multiagent Systems (MAS), specifically focusing on the problem of Coalition Structure Generation (CSG). CSG is a NP-Hard problem consisting in partitioning agents into coalitions to optimize collective performance. Our study makes three contributions: (i) a novel action space representation tailored for CSG, (ii) a comprehensive comparative analysis of multiple algorithms, and the introduction of Lazy NMCS, (iii) a cutting-edge method that surpasses previous benchmarks. By outlining efficient coalition formation strategies, our findings offer insights for advancing MAS research and practical applications.

1 INTRODUCTION

Multiagent Systems (MAS) is a vast field of study where multiple entities have different preferences, goals, or beliefs (Shoham and Leyton-Brown, 2008). One of the main goals of MAS research is to plan and coordinate agents in order to improve global performance or to complete task goals that are difficult or impossible for an individual agent.

Among the different fields of study in MAS, our work focuses on the partitioning of the agents into mutually disjoint coalitions (Rahwan et al., 2015). Partitioning agents into a coalition structure's goal can be stability (*i.e.,* where no agent has an interest in changing coalition) (Cechlárová et al., 2001) or optimality (*i.e.,* maximizing the total performance / social welfare) (Aziz and de Keijzer, 2011). Here we decide to focus on maximizing the sum of the performances of all the coalitions in the coalition structure, which is also called Coalition Structure Generation (CSG) (Rahwan et al., 2015).

Out of the existing methods used on the resolution of the CSG problem, some of them are trying to resolve optimally such as dynamic programming (Yun Yeh, 1986) or integer partition-based search (Rahwan et al., 2009). Nevertheless finding the best coalition structure, especially with many agents, will be costly since the problem is NP-complete. Therefore, methods have been introduced to produce coalition structures with better values on large number of agents at the cost of a loss in theoretical guarantees. Genetic algorithms (Sen and Dutta, 2000) and GRASP (Mauro et al., 2010) algorithms fall into this category.

In this paper we compare multiple Monte Carlo search algorithms, including the state of the art one on the CSG problem: CSG-UCT (Wu and Ramchurn, 2020). Monte Carlo search algorithms are the state of the art in many applications and have recently been combined with reinforcement algorithms, beating human professional players in multiple games such as Go, Chess, and Shogi (Silver et al., 2017; Silver et al., 2018).

Monte Carlo search algorithms are used on coalition problems in two resources. One in (Wu and Ramchurn, 2020) which uses a modified version of Upper Confidence bounds applied to Trees (UCT) (Browne et al., 2012) with a greedy playout. Another one is presented in (Präntare et al., 2021), where different Monte Carlo Search algorithms are outperformed by the Random Hill Climbing (RHC) algorithm in the Simultaneous Coalition Structure Generation and Assignment (SCSGA) (Präntare and Heintz, 2020) problem. It is stated that the SCSGA problem is an extension of the CSG problem with the inclusion of an as-

[a] https://orcid.org/0000-0002-7794-5614
[b] https://orcid.org/0000-0002-0082-1939
[c] https://orcid.org/0000-0001-9805-8291
[d] https://orcid.org/0000-0003-4669-9374

signment problem and that the RHC should perform well on the CSG problem (theorem 1 of (Präntare and Heintz, 2020)).

In this paper, we extend the research on Monte Carlo algorithms for the CSG problem by using other Monte Carlo based algorithms, either already present in the CSG literature (RHC, CSG-UCT), or new to the problem but well known (NMCS, UCT) or completely new (LNMCS). Algorithms based on NMCS showed great result in puzzles and optimization problems, particularly in multiple applications such as Single Player General Game Playing (Méhat and Cazenave, 2010), Cooperative Pathfinding (Bouzy, 2014), Software testing (Poulding and Feldt, 2014), heuristic Model-Checking (Poulding and Feldt, 2015), Games (Cazenave et al., 2016), RNA Inverse Folding problem (Portela, 2018; Cazenave and Fournier, 2020), Graph Coloring (Cazenave et al., 2020) and refutation of spectral graph theory conjectures (Roucairol and Cazenave, 2022).

We contribute to the CSG problem in three ways:

(i) We provide a new representation of the action space of the CSG problem, which can improve the performance under given conditions. (ii) We use it for the first time and compare the performance of multiple algorithms on the CSG problem. (iii) We introduce a new algorithm, the Lazy NMCS, which solves past problems of NMCS and outperforms the previous state of the art (at least) on the main benchmarks of the problem.

The paper is structured as follows: the second section presents notations for CSG problems, section three presents the various representations used, section four presents the different algorithms, section five presents our results on multiple benchmarks, and the last section summarizes our work and outlines future work.

2 CSG Model

The modelization of the action space is a key factor for the performances. One of the first model proposed was (Sandholm et al., 1999) which represents the coalition with levels, where at level i, each node is a coalition structure composed of i coalitions. This model is explained more precisely in Subsection 2.1.

Other models are available such as in (Rahwan et al., 2007b), where coalition structures are regrouped by multiset of positive integers whose sum is equal to $|A|$. This representation has been used for integer partition graph (Rahwan et al., 2009).

In Subsection 2.2, we introduce a new model that allows us to reduce the number of actions at each node and to enhance the performance under certain conditions.

2.1 Model A: simple coalition merging

The initial state is the singleton coalition (a CS composed of the $|A|$ singleton coalitions), and the available moves consist in the $|CS| \times (|CS|+1)/2$ two by two merging of coalitions among the coalition structure CS. Thus, this action space is a directed graph where each node represents a coalition structure. The graph representing the action space is therefore composed of levels, where each level corresponds to the number of coalitions in each coalition structure *i.e.,* in the level i, each node is composed of i coalitions. The graph naturally ends up with the structure made of one coalition encompassing all agents, called the grand coalition.

For an example of the CS graph with 4 agents see figure 1. In this model, the action space and the search space increase greatly with each new agent. For each node (coalition structure CS), there are $\frac{|CS|\times(|CS|-1)}{2}$ possible actions, and the closer we are from the starting node, the more actions are possible with the first one having $|CS| = |A|$. To reduce the size of the action space (4950 available moves from the singleton coalition structure with 100 agents), we introduce a new representation.

2.2 Model B : Locked Merge

In model A, all sequences of actions (playouts) lead to the grand coalition. In Monte Carlo Search algorithm, the playout usually returns the value of the last state of the playout, but in model A it will return the grand coalition value each time. To alleviate this problem it is possible to modify the playout algorithm to keep in memory the best state encountered yet and return it at the end of the playout, this is the method used with CSG-UCT (Wu and Ramchurn, 2020), however, computing the score after each move can be costly. As stated before, the action space for larger CS in model A can get large enough for it to be problematic (4950 moves for 100 coalitions).

Our aim with this new model is (i) to reduce the number of available moves, especially from the first and largest coalition structure (singleton), and (ii) to avoid the costly computation of each state's score of a playout that model A requires.

We propose a new model representation where we get a tree of the state space of depth $|A|$, with $|CS|$ moves possible at each node-state and with $|CS| = |A|$ moves for the starting node.

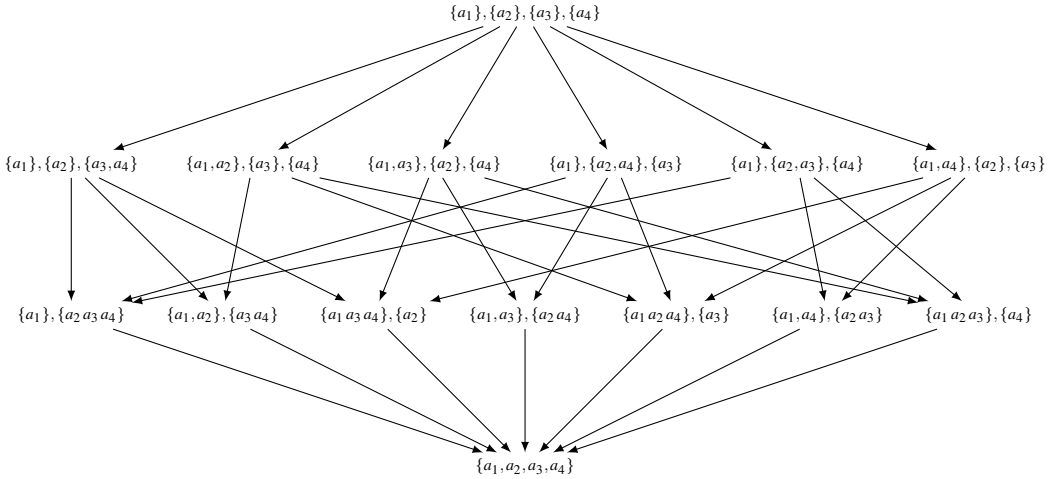


Figure 1: Model A: an example with four agents.

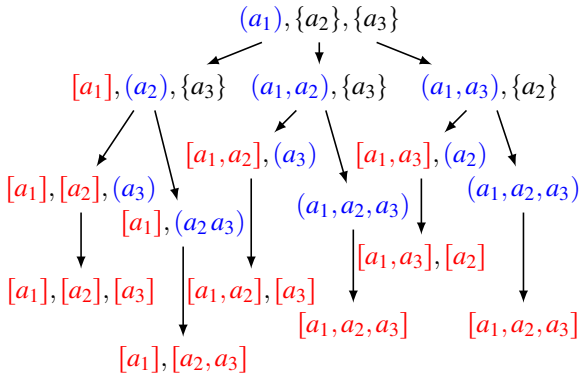


Figure 2: Model B: an example with three agents. We denote {} when the coalition is not locked and not active, () when the coalition is not locked and active, and [] when the coalition is locked.

The new model is defined as follows: The starting node is the coalition structure of all singleton coalitions as with model A, without any coalition locked (a coalition that cannot be merged and will be present as is in the final state). At any time, only one coalition is active. Two types of moves can be applied to the coalition structure: (i) locking the active coalition and selecting another coalition as the active coalition or (ii) merging another coalition with the active coalition (it remains the active coalition).

Thus any CS has exactly as many moves available as non-locked coalitions with model B, and each move played reduces the total of non-locked coalitions in the CS by 1. Once all coalitions are locked there is no more action available and it is then possible to compute the value of the coalition structure.

An example is provided in Figure 2 with three agents. Locked coalitions are noted [] and unlocked coalition as {}. As said previously, the first node is the CS of all singleton coalitions ($\{(a_1),\{a_2\},\{a_3\}\}$), with none of them being locked. From this node-state, there are three actions/moves possible, the first one is to lock the first coalition ($[a_1]$), the second action is to merge the first and second coalition ((a_1,a_2)) and the last action is to merge the first and third coalition ((a_1,a_3)). If we chose the second action, we now have two actions available. The first action is to lock the current coalition ($[a_1,a_2]$) and the second one is to merge the remaining coalitions ((a_1,a_2,a_3)). If we decide to lock the coalition, we are left with one non-locked coalition and the last action is to lock it ($[a_1,a_2],[a_3]$).

It should be noted that it is possible to modify model B to make all terminal states return different structures by not merging the coalitions that were rejected by the current active coalition until the current active coalition is locked. This results in an unbalanced tree. We did not explore this version of the model due to poor preliminary results.

3 Algorithms

In this section, we present the algorithms we tried on the CSG problem: (i) Upper Confidence bound applied to Trees (UCT) (Browne et al., 2012) (ii) CSG-UCT (Wu and Ramchurn, 2020) (iii) Random Hill Climbing (RHC) (Prántare and Heintz, 2020) (iv) Nested Monted Carlo Search (NMCS) (Cazenave, 2009) (v) Lazy Nested Monte Carlo Search (LNMCS).

In the subsequent pseudo-codes, we use the following notations:

c-st denotes the current state,

n-st denotes the next state,

b-score denotes the best score,

M_{state} denotes the legal actions possible in state,

σ denotes the sequence kept in memory,

σ^* denotes the best sequence,

b denotes the number of times we can repeat the playout algorithm.

$play(state, move)$ is a function returning the next state when *move* is applied on *state*,

l denotes the current level in NMCS and LNMCS,

$list[i..]$ denotes the part of *list* from the i-th element to the end.

3.1 Monte Carlo Tree Search and UCT

Monte Carlo Tree Search (MCTS) (Browne et al., 2012) is a popular category of tree search algorithms, notably used in recent and world-leading research projects such as Alphazero (Silver et al., 2017), Alphafold (Jumper et al., 2021) or Astrazeneca's tool for retrosynthesis AiZynthFinder (Genheden et al., 2020). MCTS consists of four steps: (i) **selection** — select nodes by going down the tree according to the exploitation policy until an unexplored node or a final state is hit (ii) **expansion** — unless the node is a terminal state, add it to the explored tree (iii) **simulation** — estimate the child node by using an exploration strategy (playout) (iv) **backpropagation** — backpropagate the result obtained from the playout through the nodes chosen during the selection phase.

3.1.1 Selection

Most of the time, the selection phase is done by bandit algorithms. Bandit algorithms are a class of algorithms used when one needs to choose between K actions. To do so, bandit algorithms must balance between the exploitation of the current best action and the exploration of other actions that are currently suboptimal.

The formula for UCT is as follow:

$$UCT_{child} = \bar{X}_{child} + C\sqrt{\frac{ln(n)}{n_{child}}}$$

The child node selected from a current node is the one that maximizes UCT_{child}. $\bar{X}_{child}$ is the average reward of the *child*, C is a constant parameter, n_{child} the number of times the child node has been visited and n the number of times the current node has been visited.

3.1.2 Simulation

In this paper, we are using two types of playouts: (i) random playout or (ii) greedy playout. Random playouts select uniformly a child node, greedy playouts select the child node with the best value (a node is a coalition structure).

Algorithm 1: Playout algorithm.

```
Function Playout(state):
    b-score ← −∞;
    σ* ← ();
    c-st ← state;
    σ ← ();
    while c-st is not terminal do
        if greedy then
            move ← Greedy(M_{c-st}) ;
        else move ← Random(M_{c-st}) ;
        c-st ← play(c-st, move);
        σ.push(move);
        if b-score ⩽ c-st.score or
          classicPlayout then
            b-score ← c-st.score;
            σ* ← σ;
    return b-score, σ*;
```

In Algorithm 1, we present the pseudo-code of the playouts used in the multiple algorithms presented later in the paper. If *classicPlayout* is true, the algorithm returns the terminal value and not the best it encountered on its path (suitable for model B).

3.1.3 Backpropagation

Once a value is obtained from the simulation step, all nodes selected during the selection step (a path going down the CS tree) see their total number of visits increased by 1, and their average reward updated with the value from the simulation.

3.2 CSG UCT

CSG-UCT is introduced in (Wu and Ramchurn, 2020) and designed for model A (Subsection 2.1). CSG-UCT differs from UCT in three ways: (i) in the selection phase, the average value of X_{child} is replaced with the maximum value observed (ii) the value backpropagated is the maximum value between the value backpropagated and the current value saved. (iii) The playouts are greedy, thus CSG-UCT cannot work for model B.

Greedy playouts do not select the next state uniformly like random playouts, instead, they select the state (merge the two coalitions C_1 and C_2) that will improve the coalition structure value the most: $argmax_{C_1,C_2 \in CS} v(\{C_1 \cup C_2\}) - v(C_1) - v(C_2)$.

3.3 Random Hill Climbing

Random Hill Climbing (RHC) is defined in (Präntare and Heintz, 2020). In this work, they compare a basic version of MCTS against RHC and obtain better results with RHC. The authors compare the algorithms over the Simultaneous Coalition Structure Generation and Assignment (SCSGA) problem, which is an extension of CSG with an assignment problem. They claim that an algorithm that can provide good results on an instance of the SCSGA problem can also provide good results on a CSG instance, so we decided to compare RHC against the other algorithms.

Algorithm 2: RHC algorithm.

```
Function RHC(b):
    b-st ← RandomCoalitionStructure() ;
    while b not exhausted do
        CS ← RandomCoalitionStructure() ;
        succes ← true;
        while succes == true and b not
          exhausted do
            success ← false;
            for a in a_{k_1}, ..., a_{k_n} do
                i ← l such that a ∈ C_l ;
                i* ←
                  argmax_{j∈{1,...,m}\{i}} Δ_a(C_j) ;
                if
                  Δ_a(C_{i*}\{a})) > Δ_a(C_i\{a}))
                  then
                    success ← true;
                    CS[i] ← CS[i]\{a};
                    CS[i*] ← CS[i*] ∪ {a};
        if b-st.score > CS.score then
            b-st ← CS ;
    return b-st
```

RHC uses neither of the models (A or B). Instead, RHC starts from a randomly generated CS and for each agent checks if swapping with any coalition would increase the value of the CS, if so the agent swaps coalitions with the one providing the largest marginal contribution. If none of the agents swapped to another coalition, the value is returned as a potential optimal CS, and RHC is restarted from another random CS until the budget b is exhausted. The pseudo-code of RHC is available in Algorithm 2, and has been modified to match the CSG formalism. $\Delta_a(C) = v(C \cup a) - v(C)$ is the marginal contribution of agent a to the coalition C.

Algorithm 3: NMCS algorithm.

```
Function nmcs(c-st, l):
    if l = 0 then return Playout(c-st) ;
    b-score ← −∞;
    σ* ← [];
    ply ← 0;
    while c-st is not terminal do
        foreach move in M_{c-st} do
            n-st ← play(c-st, move);
            (score, σ) ←nmcs ( n-st, l − 1 );
            if score ⩾ b-score then
                b-score ← score;
                σ*[ply..] ← move + σ;
        next − move ← σ*[ply];
        ply ← ply + 1;
        c-st ← play(c-st, next − move);
    return b-score, σ*
```

3.4 NMCS

Nested Monte Carlo Search (NMCS) (Cazenave, 2009) is a Monte Carlo Search algorithm that recursively calls a lower level of NMCS on each child state of the current state. This lower level of NMCS allows the algorithm to decide which move to choose next. The lowest level of NMCS being a random playout. The main improvement of NMCS is the memorization of the best sequence at each recursion level.

NMCS is available in Algorithm 3 and in all our experiments with NMCS we used a level l of 3.

3.5 LNMCS

The Lazy NMCS inherits its main features from the NMCS, but solves an obstacle encountered for the CSG problem. Calling a higher level NMCS ($l \geqslant 3$) yields better results. However, the cost of calling a lower level $l-1$ NMCS on each of the resulting states of the available actions can be prohibitive and some of these actions produce subtrees doomed to produce underwhelming results.

Therefore, we propose a new algorithm based on NMCS named Lazy NMCS (LNMCS). LNMCS was first proposed as a prototype and applied to the HP-model for protein folding (Roucairol and Cazenave, 2023), this new version corrects some flaws of the prototype such as the separation between evaluation and pruning. LNMCS works the same as NMCS with the following exceptions (i) before expanding a state, we compute the mean of each available action by launching b playouts (ii) we update a dynamic threshold relative to the depth of the current state (iii) we compare the score of each child to the threshold, if

the score is below the threshold, the node is pruned.

The pseudocode of LNMCS is available in Algorithm 4 and you can find each part of this process marked in the pseudocode.

In addition to past notation, we are using r as the ratio to the threshold a state will be pruned, e is the number of possible moves we will focus on in case there are too many moves, and, as in NMCS, l is the nesting level. tr is a list of tuples containing the mean value and the number of experiments made to contribute to that value in order to compute the mean easily. $trmax$ keeps in memory the best evaluation for each level of depth. $randomSample(M_{state}, e)$ randomly selects e actions from the moves from $state$ if there is too many available actions.

See Figure 3 for a graphic description of LNMCS, subtrees are sampled and the underperforming ones are pruned.

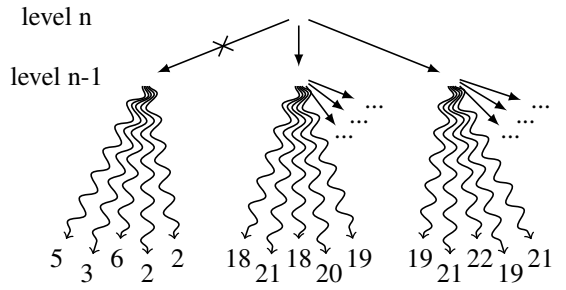


Figure 3: Level n LNMCS pruning a search subtree and launching n-1 LNMCS on surviving search subtrees.

4 Results

4.1 Experimental setup

To refer to our models-algorithms combinations, we use the following notations:

C_A: model A CSG-UCT, $C = 1$

L_A: model A LNMCS, $r = 0$, $b = 2$, $l = 5$, $e = 10$

N_A: model A NMCS, $l = 3$

U_A: model A UCT, $C = 1$

L_B: model B LNMCS, $r = 0$, $b = 2$, $l = 5$, $e = 10$

F_B: (Full action space) model B LNMCS, $r = 0.9$, $b = 2$, $l = 5$, $e = 100$

N_B: model B NMCS, $l = 3$

U_B: model B UCT, $C = 1$

L_G: model A LNMCS with greedy playouts, $r = 0$, $b = 1$, $l = 5$, $e = 10$

N_G: model A NMCS with greedy playouts, $l = 3$

R: RHC

Algorithm 4: LNMCS algorithm.

```
tr ← [];
trmax ← [];
Function lnmcs(c-st, l, b, r, e):
    if l = 0 then return Playout(c-st) ;
    b-score ← −∞;
    σ* ← [];
    ply ← 0;
    while c-st is not terminal do
        budget_moves ←
          randomSample(M_c-state, e)
        candidates ← [];
        d ← c-st.nbplay;
        /* d: number of moves played
           from initial state        */
        foreach move in budget_moves do
            n-st ← play(c-st, move);
            ev ← 0.0;
            /* (i)                   */
            for _ in 0..b do
                (plsc, plsq) ←
                  Playout(n-st);
                if score ⩾ b-score then
                    b-score ← plsc;
                    σ*[ply..] ← move + plsq;
                ev ← ev + plsc;
                n ← n + 1;
            candidates.push([ev, move]);
            /* (ii)                  */
            if tr.length() < d + 1 then
                tr.push(val : 0.0, n : 0);
                trmax.push(ev);
            tr[d].val ← (tr[d].val∗tr[d].n+ev) / (tr[d].n+1);
            tr[d].n ← (tr[d].n + 1);
            if trmax[d] < ev/n then
              trmax[d] ← ev/n ;
        /* (iii)                     */
        foreach can in candidates do
            nl ← l − 1;
            if can[0] <
              tr[d] + r · (trmax[d] − tr[d])
              then nl ← 0 ;
            (score, σ) ←
              lnmcs(play(c-st, can[1]), nl, p) ;
            if score ⩾ b-score then
                b-score ← score;
                σ*[ply..] ← can[1] + σ;
        next-move ← σ*[ply];
        ply ← ply + 1;
        c-st ← play(c-st, next-move)
    return b-score, σ*;
```

To compare these algorithms, we launched 100 instances of the CSG problem with 100 agents with a time budget of 100 seconds on four benchmarks.

As the CS values of an instance of the problem are randomly initialized, we decide to compare the result by measuring the number of times an algorithm is better than another on each of the 100 instances.

The average performances and the standard deviation are vulnerable to differences among the 100 different synthetic problem instances we used *i.e.,* when the standard deviation does not go below 0.5 on the gaussian benchmark, it is in part due to the optimal structure score having a standard deviation of about 0.5 over the 100 instances.

We chose to compare our algorithms on four coalition value distributions/benchmarks from the literature:

- **Uniform** first used in (Larson and Sandholm, 1999) *i.e.,* $v(C) \sim \mathcal{U}(0,|C|)$.
- **Normal** or **Gaussian** first used in (Rahwan et al., 2007a) *i.e.,* $v(C) \sim \mathcal{N}(10*|C|,0.1)$, $\sigma = 0.1$ being the standard deviation.
- **Agent based** first used in (Rahwan et al., 2021) *i.e.,* $v(C) \sim \sum_{a \in C} \mathcal{U}(0,p_a)$ where $p_a \sim \mathcal{U}(0,1)$ is the power of an agent and is fixed on start..
- **NDCS** first used in (Rahwan et al., 2009) *i.e.,* $v(C) \sim \mathcal{N}(|C|,\sqrt{|C|})$, $\sigma = \sqrt{|C|}$ being the standard deviation.

The experiments were made with Rust 1.59, on an Intel Core i7-11850H 2.50GHz using a single core (but parallel processing is very accessible). We use a random generator with a set seed as our value function, and the values of each coalition are only produced once on demand by the random generator and then stored in a hashmap for later use. The raw results are available in Table 1.

4.2 Raw results

As observed in Table 1, on the uniform benchmark, the LNMCS with model B significantly outperforms all of the other algorithms, with the greedy LNMCS coming in second place. Surprisingly, CSG-UCT did not perform very well and was only able to outperform UCTs and NMCS. On the Gaussian, NDCS, and agent-based benchmarks, the difference is even greater with the greedy LNMCS, being close to 100 wins each time against each of the other algorithms.

By calculating confidence intervals, we can assert with a confidence of 95% that one method is superior to another only if that method wins at least 60 times out of 100, and with a confidence of 99% if it wins at least 63 times. LNMCS outperforms other methods significantly. The only duel that would leave any doubts about the performances of the LNMCS is the one between the greedy LNMCS and CSG-UCT on the Gaussian benchmark. We decided to run 100 additional experiments (seeds 100 to 199), and obtained 62 wins for the greedy LNMCS and 38 for CSG-UCT. These experiments give a 0.99% certitude that the A LNMCS is at least slightly superior to the A CSG-UCT on the Gaussian benchmark. We think this performance can be explained by the fixed variance of the Gaussian coalition value function and does not favor larger coalitions. Since UCT explores from the root every time it is advantaged at finding small but high-value coalitions.

In the next sections, we analyze the performances of each algorithm relative to the others and explain these results.

4.2.1 Playout choice

MCTS/UCT generally uses a random playout, however, the CSG-UCT algorithm uses a strictly greedy playout. The authors of CSG-UCT (Wu and Ramchurn, 2020) did not compare the impact of using a different playout. We propose to look into the effects of the playout type, both for UCT and for the other algorithms.

By looking at the results from the uniform benchmark in Table 1 (a), we can observe that C_A outperforms U_A (CSG-UCT is comparable to UCT with greedy playouts) with 82 wins, L_G performs better than L_A with 60 wins and N_G performs better than N_A with 61 wins.

On the Gaussian, NDCS, and agent-based benchmarks, the results show that the performance of the greedy is further enhanced, to such an extent that the greedy playout does not lose a single time against a random playout.

While the greedy playouts seem more effective, retrieving the values of all the possible child CS (up to 4950 with model A) can be costly and slows down the playouts. It is the most resource-consuming part of all of these algorithms.

4.2.2 Model choice

Model B (random playouts only) provides superior results on the uniform benchmark with the LNMCS, being able to outperform the LNMCS on model A with greedy playouts and with random playouts. However, it provides far inferior results on the other benchmark. We think it's due to the other benchmarks favoring trying as many coalitions as possible, which model B can not do since it only returns the terminal CS. It

Table 1: The uncurated data, number of times the algorithm from a line beats the algorithm from a column over 100 experiments.

	C_A	L_A	N_A	U_A	L_B	F_B	N_B	U_B	L_G	N_G	R	total wins	Copeland
C_A	/	36	53	82	6	24	97	74	13	39	100	524	538.5
L_A	63	/	60	83	22	45	97	81	39	65	100	655	661.5
N_A	46	38	/	67	16	30	97	73	24	39	100	530	537.5
U_A	18	16	32	/	3	14	89	46	9	16	100	343	347.5
L_B	**94**	**76**	**83**	**95**	/	**77**	**99**	**94**	**74**	**92**	**100**	884	892
F_B	74	53	66	83	16	/	97	84	43	64	100	680	690
N_B	3	3	3	11	1	3	/	8	1	1	100	134	134
U_B	24	15	24	48	4	12	92	/	8	24	100	351	361.5
L_G	81	60	75	91	26	55	99	90	/	77	100	754	767
N_G	48	35	61	84	8	33	99	75	16	/	100	559	570.5
R	0	0	0	0	0	0	0	0	0	0	/	0	0

(a) Uniform benchmark with 100 agents.

	C_A	L_A	N_A	U_A	L_B	F_B	N_B	U_B	L_G	N_G	R	total wins	Copeland
C_A	/	100	100	100	100	100	100	100	36	94	100	930	930
L_A	0	/	40	100	100	100	100	100	0	0	9	549	549
N_A	0	60	/	99	99	99	99	100	0	0	27	583	583
U_A	0	0	1	/	86	38	100	100	0	0	0	225	325
L_B	0	0	1	14	/	1	99	100	0	0	0	215	215
F_B	0	0	1	62	99	/	100	100	0	0	0	362	362
N_B	0	0	1	0	1	0	/	100	0	0	0	102	102
U_B	0	0	0	0	0	0	0	/	0	0	0	0	0
L_G	**64**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	/	**94**	**100**	958	958
N_G	6	100	100	100	100	100	100	100	6	/	100	812	812
R	0	91	73	100	100	100	100	100	0	0	/	664	664

(b) Gaussian benchmark with 100 agents.

	C_A	L_A	N_A	U_A	L_B	F_B	N_B	U_B	L_G	N_G	R	total wins	Copeland
C_A	/	100	100	100	100	100	100	100	0	73	100	873	873
L_A	0	/	84	100	97	93	100	100	0	0	3	577	577
N_A	0	16	/	92	39	31	48	94	0	0	4	324	324
U_A	0	0	8	/	0	0	0	45	0	0	0	53	53
L_B	0	3	61	100	/	39	82	100	0	0	1	386	386
F_B	0	7	69	100	61	/	94	100	0	0	2	433	433
N_B	0	0	52	100	18	6	/	100	0	0	1	277	277
U_B	0	0	6	55	0	0	0	/	0	0	0	61	61
L_G	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	/	**100**	**100**	1000	1000
N_G	27	100	100	100	100	100	100	100	0	/	100	827	827
R	0	97	96	100	99	98	99	100	0	0	/	689	689

(c) Agent based benchmark with 100 agents.

	C_A	L_A	N_A	U_A	L_B	F_B	N_B	U_B	L_G	N_G	R	total wins	Copeland
C_A	/	100	100	100	100	100	100	100	0	85	100	885	885
L_A	0	/	79	100	98	87	100	100	0	0	2	566	566
N_A	0	21	/	95	37	33	43	94	0	0	2	325	325
U_A	0	0	5	/	0	0	0	53	0	0	0	58	58
L_B	0	2	63	100	/	33	91	100	0	0	0	389	389
F_B	0	13	67	100	67	/	94	100	0	0	1	442	442
N_B	0	0	57	100	8	6	/	100	0	0	1	272	272
U_B	0	0	6	47	0	0	0	/	0	0	0	53	53
L_G	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	/	**100**	**100**	1000	1000
N_G	15	100	100	100	100	100	100	100	0	/	100	815	815
R	0	98	98	100	100	99	100	100	0	0	/	695	695

(d) NDCS benchmark with 100 agents.

For example, L_B beats N_A 83 times out of 100 with 1 ex-aequo on the uniform benchmark, and F_B beats U_A 62 times out of 100 with no ex-aequo on the gaussian benchmark.

however proves the interest of trying new representations of the problem.

4.2.3 Algorithm family choice

With regard to the choice of the type of algorithm, the nested family is overall preferable to the MCTS family on the CSG problem, especially with LNMCS which dominates in every benchmark. From the MCTS family, we observe overall great performances with the CGT-UCT except on the uniform benchmark.

Precisely, we observe the following dominance orders:

Uniform: $L_B \succ L_G \succ F_B \succ L_A \succ C_A \succ others$
Gaussian: $L_G \succ C_A \succ N_G \succ R \succ N_A \succ others$
Agent-based: $L_G \succ C_A \succ N_G \succ R \succ L_A \succ others$
NDCS: $L_G \succ C_A \succ N_G \succ R \succ L_A \succ others$

4.2.4 Discussion: The benchmark problem

As you can see in Table 1 (b, c, d), and in the previous observations, most random playout based algorithms perform poorly compared to their greedy playout-based versions on the Gaussian, agent-based, and NDCS benchmarks. We can notice that this is not the case for Sandholm's initial uniform benchmark.

To understand why, we computed the optimal coalitions for instances of the problem with 15 agents using an exact algorithm, and then compared the CSG-UCT to a BEAM search with a width of 10. On such small instances of the problem, the BEAM search returned the optimal value, slightly higher or equal to the value returned by CSG-UCT.

We tried various other benchmarks such as Sandholm's second uniform benchmark, where the value of a coalition is sampled uniformly between 0 and 1 regardless of its size (Sandholm et al., 1999). It turns out that every benchmark other than Sandholm's first uniform greatly favors greedy playouts and gives similar results to the Gaussian benchmark. It is the main reason why we decided to experiment on only 3 of the benchmarks introduced by Rahwan.

Alternatively, the RHC algorithm which consists of a greedy playout stopping at the first local maximum and starting from a randomly initialized state outperforms the random playouts-based state of the art machine learning algorithms on the agent-based, NDCS, and Gaussian distributions. For these distributions, a single greedy playout is much better than algorithms using random playouts. That result leads us to question the interest of these distribution, and others introduced by Rahwan, as their introduction was never justified in the first place and their number is getting out of hand.

As shown in Table 1, replacing LNMCS random playouts with greedy playouts is enough to outperform the current state-of-the-art algorithms.

5 Conclusion and Future Works

In this paper, we proposed to analyze Nested Monte Carlo based algorithms for the CSG problem. We present a new algorithm called Lazy Nested Monte Carlo Search which answers some of NMCS's shortcomings. In addition, we present a new model representation of CSG which allows us to strongly reduce the number of actions at the beginning of the search.

Our new algorithm is able to outperform the previous state-of-the-art algorithms on all of the main coalition value distributions we experienced upon. We also proposed a new modelization of the search tree that provides better results over the initial uniform distribution.

In future works, we may aim at:

(i) Finding real-life coalition value distributions to compare algorithms on real problems. In this work, we have been assuming that coalition values are not affected by other coalitions. In many realistic settings, such as in the Partition Function Games (PFG) formalism (Thrall and Lucas, 1963), this property is not satisfied. Another task will be to extend our work to probabilistic CSG (Schwind et al., 2021).

(ii) Proposing a new coalition value distribution that is resistant to the greedy playouts approach to further the CSG problem.

You can access our implementation as well as the result files containing the value improvements and their timestamps at https://github.com/RoucairolMilo/coalition.

REFERENCES

Aziz, H. and de Keijzer, B. (2011). Complexity of coalition structure generation.

Bouzy, B. (2014). Monte-Carlo Fork Search for Cooperative Path-Finding. In Cazenave, T., Winands, M. H., and Iida, H., editors, *Computer Games*, pages 1–15, Cham. Springer International Publishing.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A Survey of Monte Carlo Tree Search Methods. *IEEE Trans. Comput. Intell. AI Games*, 4(1):1–43.

Cazenave, T. (2009). Nested Monte-Carlo Search. In Boutilier, C., editor, *IJCAI*, pages 456–461.

Cazenave, T. and Fournier, T. (2020). Monte Carlo inverse folding. In *Monte Carlo Search at IJCAI*.

Cazenave, T., Negrevergne, B., and Sikora, F. (2020). Monte Carlo graph coloring. In *Monte Carlo Search at IJCAI*.

Cazenave, T., Saffidine, A., Schofield, M. J., and Thielscher, M. (2016). Nested monte carlo search for two-player games. In *AAAI*, pages 687–693.

Cechlárová, K., Romero-Medina, A., et al. (2001). Stability in coalition formation games. *International Journal of Game Theory*, 29(4):487–494.

Genheden, S., Thakkar, A., Chadimová, V., Reymond, J.-L., Engkvist, O., and Bjerrum, E. (2020). AiZynthFinder: a fast, robust and flexible open-source software for retrosynthetic planning. *Journal of Cheminformatics*, 12(1):70.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S. A. A., Ballard, A. J., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., Back, T., Petersen, S., Reiman, D., Clancy, E., Zielinski, M., Steinegger, M., Pacholska, M., Berghammer, T., Bodenstein, S., Silver, D., Vinyals, O., Senior, A. W., Kavukcuoglu, K., Kohli, P., and Hassabis, D. (2021). Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589.

Larson, K. S. and Sandholm, T. W. (1999). Anytime coalition structure generation: an average case study. In *Proceedings of the third annual conference on Autonomous Agents*, pages 40–47.

Mauro, N. D., Basile, T., Ferilli, S., and Esposito, F. (2010). Coalition structure generation with grasp. In *International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, pages 111–120. Springer.

Méhat, J. and Cazenave, T. (2010). Combining UCT and Nested Monte Carlo Search for single-player general game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):271–277.

Portela, F. (2018). An unexpectedly effective Monte Carlo technique for the RNA inverse folding problem. *BioRxiv*, page 345587.

Poulding, S. M. and Feldt, R. (2014). Generating structured test data with specific properties using nested Monte-Carlo search. In *GECCO*, pages 1279–1286.

Poulding, S. M. and Feldt, R. (2015). Heuristic model checking using a Monte-Carlo tree search algorithm. In *GECCO*, pages 1359–1366.

Präntare, F., Appelgren, H., and Heintz, F. (2021). Anytime heuristic and monte carlo methods for large-scale simultaneous coalition structure generation and assignment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11317–11324.

Präntare, F. and Heintz, F. (2020). An anytime algorithm for optimal simultaneous coalition structure generation and assignment. *Autonomous Agents and Multi-Agent Systems*, 34(1):1–31.

Rahwan, T., Michalak, T., and Jennings, N. (2021). A hybrid algorithm for coalition structure generation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 26(1):1443–1449.

Rahwan, T., Michalak, T. P., Wooldridge, M., and Jennings, N. R. (2015). Coalition structure generation: A survey. *Artificial Intelligence*, 229:139–174.

Rahwan, T., Ramchurn, S. D., Dang, V. D., Giovannucci, A., and Jennings, N. R. (2007a). Anytime optimal coalition structure generation. In *AAAI*, volume 7, pages 1184–1190.

Rahwan, T., Ramchurn, S. D., Dang, V. D., and Jennings, N. R. (2007b). Near-optimal anytime coalition structure generation. In *IJCAI*, volume 7, pages 2365–2371.

Rahwan, T., Ramchurn, S. D., Jennings, N. R., and Giovannucci, A. (2009). An anytime algorithm for optimal coalition structure generation. *Journal of artificial intelligence research*, 34:521–567.

Roucairol, M. and Cazenave, T. (2022). Refutation of spectral graph theory conjectures with monte carlo search. In *COCOON 2022*.

Roucairol, M. and Cazenave, T. (2023). Solving the hydrophobic-polar model with nested monte carlo search. In *International Conference on Computational Collective Intelligence*, pages 619–631. Springer.

Sandholm, T., Larson, K., Andersson, M., Shehory, O., and Tohmé, F. (1999). Coalition structure generation with worst case guarantees. *Artificial intelligence*, 111(1-2):209–238.

Schwind, N., Okimoto, T., Inoue, K., Hirayama, K., Lagniez, J.-M., and Marquis, P. (2021). On the computation of probabilistic coalition structures. *Autonomous Agents and Multi-Agent Systems*, 35(1):1–38.

Sen, S. and Dutta, P. S. (2000). Searching for optimal coalition structures. In *Proceedings Fourth International Conference on MultiAgent Systems*, pages 287–292. IEEE.

Shoham, Y. and Leyton-Brown, K. (2008). *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T. P., Simonyan, K., and Hassabis, D. (2017). Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *ArXiv*, abs/1712.01815.

Thrall, R. M. and Lucas, W. F. (1963). N-person games in partition function form. *Naval Research Logistics Quarterly*, 10:281–298.

Wu, F. and Ramchurn, S. D. (2020). Monte-carlo tree search for scalable coalition formation. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 407–413, Yokohama, Japan.

Yun Yeh, D. (1986). A dynamic programming approach to the complete set partitioning problem. *BIT Numerical Mathematics*, 26(4):467–474.

Chapter 3

Cheminformatics

3.1 Hydrophobic Polar Model

This problem was placed in the Cheminformatics section, but it is a typical computer scientist NP-hard problem. It is meant as an abstraction for protein folding where a string is folded in a 3D lattice. In that regard it is very similar to toy puzzles like the rubik's snake. As such, it was quite fun to work with, and we recommend it to any computer scientist, no chemistry knowledge is needed.

One particularity of the HP-model search space is the commitment. A move earlier in the search tree leads to greatly different states down the tree. The previous models involve less commitment: two coalitions can be merged later, the order in which the edges are added does not matter.

But with the HP-model, a sub-optimal move at one point can doom the rest of the search tree. The NMCS can be vulnerable to these wrong moves as it calls a lower level NMCS on each child. This is how we got the idea for the LNMCS: the NMCS needed a way to tell if a move was bad before launching a lower-level NMCS and spending computation time on it. This led to an algorithm providing significant improvements over the base NMCS.

While our work on the HP-model may not help chemists much, or be impressive on its own, it was the origin of one of the main contributions of this thesis, the LNMCS. The LNMCS has then showed to outperform all other MCTS algorithms on every problem it was tried on.

A few information about the LNMCS are absent from this paper, as it was the first. They are also absent from most subsequent papers as it only became clear after trying it on multiple projects. If you want to use the LNMCS, do it with a search space where mistakes are possible. The more the mistakes doom a branch of the search tree, the more the LNMCS will outperform the NMCS. For example, it is easy to course correct when generating graphs due to isomorphisms and the existence of multiple paths to a single graph, the same for the nonograms where a move can undo the

previous move. On the other hand, the HP-model has the search space that is the most prone to doom a subtree in all the projects of this thesis, subsequent moves can't save the subtree, and a wrong move can make the value of the state decrease significantly more than when generating graphs. The LNMCS should however not underperform compared to the NMCS on any search space: it would mean that the evaluation of a state is useless, and because it is made with playouts or a heuristic, most MCTS algorithms should struggle equally then. Another tip to get the best out of the LNMCS is to ensure that at least 50% of the states are pruned (improves more than twice the performances of the algorithm because many pruned states are the root of costly branches, higher level LNMCS). The ratio varies from search space to search space and the hyperparameters must be tuned, but it is a quick task.

Solving the Hydrophobic-Polar model with Nested Monte Carlo Search

Milo Roucairol and Tristan Cazenave

LAMSADE, Université Paris Dauphine - PSL, CNRS, Paris, France

Abstract. In this paper we present a new Monte Carlo Search (MCS) algorithm for finding the ground state energy of proteins in the Hydrophobic-Polar-model (HP model). We also compare it to other MCS algorithms not usually used on the HP model as well as to other approaches and provide an overview of the state of the art algorithms used on the HP model.

Keywords: Protein folding · Monte Carlo Search · MCTS · HP model · optimization · pruning.

1 Introduction

Monte Carlo search algorithms have proven to be powerful as game playing agents, with recent successes like AlphaGo [14]. These algorithms have the advantage of only needing an evaluation function for the final state of the space they explore.

Protein folding is crucial to our understanding of biology and designing drugs, however, trying our algorithms directly on accurate models could be counter-productive. In this paper, we use a new MCS algorithm to fold proteins in a simplified lattice based model called the HP model.

First, we will present the protein folding and the HP model, then the different algorithms we used to explore the problem space and finally the results of our experiments.

2 The problem

2.1 Protein folding

With recent developments in ARNm technology, it is now possible to incite cells to produce a specific protein [8], like the spike protein used in COVID-19 vaccines. Unfortunately, deducing the shape a protein will take given the amino acids sequence is not obvious nor trivial. That is a reason protein folding is a very important problem in molecular biology and medicine.

Proteins are chains of amino acids (primary structure), they can fold in many different ways, the secondary structure is the shape the protein will take at a local level (a coil for example), the tertiary structure is the global shape of the

protein with less discernible patterns, finally, the quaternary structure is how a protein can assemble with another. Here we are interested in predicting the ground state energy folding (secondary and tertiary structure) from the primary structure. Many forces drive the folding, which prevents the creation of a very accurate simulator, the main driving force is the hydrophobic one.

One can not approach protein folding without mentioning DeepMind's AlphaFold [10]. Placing first at the Critical Assessment of Techniques for Protein Structure Prediction in 2018 and 2020, it is the best program for protein structure prediction yet. AlphaFold uses machine learning on a large protein database to train neural networks, in addition to physics based rules in order to predict the folding of a protein.

AlphaFold is the greatest achievement to protein folding prediction in decades, but the research is not over yet. AlphaFold accuracy can still be perfected, and by using neural networks the explainability is low and the model may not be able to predict the structure of proteins never seen before. Our objective here is to provide a better algorithm than the other MCS algorithms for Monte Carlo physics simulation, which may be more explainable than AlphaFold and other MCS algorithms, but should currently be way less accurate than Alphafold if it was applied on the same problem.

2.2 Hydrophobic-Polar model

The HP model was introduced in 1985 by Ken Dill [4]. The main idea behind the creation of the HP model is that the Hydrophobic-Polar (HP) force is the main force driving the folding of a protein, thus it is the only one used here.

The HP model is a simplified lattice based model for protein folding, it exists in 2D and 3D versions.

In the HP model, proteins are represented as a chain of H and P residues (amino acids), the chain is then folded onto a grid, two residues can not share the same positions. The residue contacts determine the energy of a chain, usually, the reward for an H-H connection is -1, and 0 for H-P and P-P contacts in a context of minimization (since the ground energy state is the state with the least potential energy). Other rewards can be used to obtain different results or guide the search.

2.3 State of the art on HP model

The HP model has seen a number of algorithms trying to solve it. All of the best performing algorithms on the HP model are Monte Carlo based, policy learning using neural networks or reinforcement learning like NRPA [12] led to poor results. In these Monte Carlo algorithms, we can identify two types, the chain growth algorithms and the replica exchange ones.

The chain growth methods add the residues one after the other, next to the previous one, it is similar to a self avoiding walk and it is the method we used in our own algorithm.

The replica exchange methods use pull moves, pulling the chain at one point by rotating a residue around one of its neighbors, symmetrically rotating a part of the chain or pulling from one side of the chain. This means the entirety of the chain is present on the lattice at any given state of the search and is in a physically possible conformation, this method is used in simulated annealing like Monte Carlo algorithms. To see a representation of these moves see Chris Thchuk, Alena Shmygelska, and Holger H Hoos REMC article [16].

Here is a short review of the methods we encountered.

1) PERM: Initially used on Self Avoiding Walks (SAW), PERM is a chain growth algorithm and was used on the HP model by Peter Grassberger in 1997 [7]. It stands for Pruned Enriched Rosenbluth Method, the idea is to explore the possible chains uniformly with a bias on the immediate gain, cutting (pruning) branches leading to too few choices and poor performances, and cloning (enriching) branches that lead to great results. PERM has seen many new versions until 2011 [9], mainly proposed by its creator, Peter Grassberger. It still is one of the best algorithms available but has been outperformed by pull-moves based more recent algorithms. We reproduced PERM, but our results with this algorithm did not live up to the expectations, it matched UCT's (see part 4.3) performances.

2) REMC: Introduced in 2007 by Chris Thachuk, Alena Shmygelska and Holger H Hoos [16], the Replica Exchange Monte Carlo algorithm uses pull moves and simulated annealing. That algorithm keeps only a certain number of replicas (it was determined the best number of replicas for the 3D HP model was 2), each with a given temperature. At each step the algorithm mutates each replica with a Monte-Carlo Search using the pull moves, the probabilities of keeping a mutation are decided by the score gain (energy loss) of the mutated state and the temperature. Then, once each state is produced through mutation, the replicas are then again swapped probabilistically according to their score (energy) and temperatures.

3) Wang-Landau sampling: Introduced in 2012 on the HP model, the Wang-Landau sampling (WLS) [17] method seems to be the new best algorithm for solving the HP model. It is a replica-exchange (simulated annealing) algorithm that uses the same pull moves as the REMC, but also uses moves consisting in cutting and joining of the molecule (thus reallocating all the residues according to their position), together they are named the Monte-Carlo trial moves. With these moves, the WLS explores the conformation space to estimate a histogram of the energies of these conformations. With this histogram, the WLS can then direct the exchange of the replicas.

3 Algorithm

3.1 Biased Growth

In a similar way to the PERM algorithm [9], we try to favor the immediate reward when building/folding the molecule. To do this, we use biased playouts, the chances of selecting a move m from M the possible moves from state S follows a softmax distribution with b the bias factor:

$$\frac{exp(G_m * b)}{\sum_{i \in M_S} exp(G_i * b)} \tag{1}$$

$M_{c\text{-}state}$ denotes the legal moves available from the state *current-state*. $G_{M_{c\text{-}state}}$ denotes the immediate gains of each legal moves available from the state *current-state*.

Algorithm 1 The biased growth playout algorithm.

```
1: function PLAYOUT(c-state, b)
2:     ply ← 0
3:     seq ← {}
4:     while c-state is not terminal do
5:         gains ← G_{M_{c-state}}
6:         move ← softMaxChoice(M_{c-state}, gains * b)
7:         c-state ← play(c-state, move)
8:         seq[ply] ← move
9:         ply+ = 1
10:    end while
11:     return score(c-state), seq
12: end function
```

3.2 Nested Monte Carlo Search

NMCS [1] is a Monte Carlo Search algorithm that 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 move choices, and it is the case here with the biased growth playouts.

Algorithm 2 gives the NMCS algorithm, l is the nesting level and b the playout bias.

3.3 Lazy Nested Monte Carlo Search

The lazy NMCS inherits its main features from the NMCS, but solves an obstacle encountered on this problem. Solving the 3D HP model with the NMCS requires

Algorithm 2 The NMCS algorithm.

```
 1: function NMCS(c-state, l, b)
 2:     if l = 0 then return playout(c-state, b)
 3:     else
 4:         best-score ← −∞
 5:         best-sequence ← []
 6:         ply ← 0
 7:         while c-state is not terminal do
 8:             for each move in M_{c-state} do
 9:                 n-st ← play(c-state, move)
10:                 (score, seq) ← NMCS(n-st, l − 1, b)
11:                 if score ≥ best-score then
12:                     best-score ← score
13:                     best-sequence[ply..] ← move + seq
14:                 end if
15:             end for
16:             next-move ← best-sequence[ply]
17:             ply ← ply + 1
18:             c-state ← play(c-state, next-move)
19:         end while
20:         return (best-score, best-sequence)
21:     end if
22: end function
```

using a level of 4 at least, however, it requires computing many 3 level NMCS, already very costly, one for each possible move the level 4 NMCS can make. The main idea behind the lazy NMCS is that there are moves that lead to low potential states, to do so, we estimate the potential of a state by launching a number of biased growth playouts and calculating the mean of their scores, then we compare that score to a threshold (relative to the number of moves already done) calculated from the previous estimations to decide if we want to expand the search tree from this state, or prune it. To update the pruning threshold tr, it is possible to use a mean, a median, or a max from the previous estimations, here we use the max as it gave the best results on these problems.

In the following pseudocode in algorithm 3, p is the number of playouts used to evaluate a state and r is the ratio to the threshold a state will be pruned on. l is the nesting level and b is the playout bias.

From line 9 to line 14, the state is evaluated with the mean of p playouts.

From line 15 to line 17, the threshold list is extended on the first entry of a new molecule length, this step is not needed if the list is initialized with the right size from the start for problems we know the maximum number of moves that will be played.

From lines 18 to 120, the threshold is updated with the evaluation.

From lines 21 to 25, it is decided with the evaluation, the pruning ratio, and the corresponding threshold if the search will be costly or not.

Algorithm 3 The Lazy NMCS algorithm.

```
1: tr ← []
2: function LNMCS(c-st, l, b, p, r)
3:     if level = 0 then return PLAYOUT(c-st, b)
4:     else
5:         best-score ← −∞
6:         best-sq ← []
7:         ply ← 0
8:         while c-st is not terminal do
9:             for each move in M_{c-state} do
10:                n-st ← play(c-st, move)
11:                for i in 0..p do
12:                    (playoutSc, _) ← playout(n-st, b)
13:                    es ← es + playoutSc/p
14:                end for
15:                if tr.length() < c-st.nbplay + 1 then
16:                    tr.push(0.0)
17:                end if
18:                if tr[c-state.nbplay] < es then
19:                    tr[c-state.nbplay] ← es
20:                end if
21:                if es < ratio ∗ tr[c-st.nbplay] then
22:                    (sc, sq) ←LNMCS(c.1, 0, b, p, r)
23:                else
24:                    (sc, sq) ←LNMCS(c.1, l − 1, b, p, r)
25:                end if
26:                if sc ≥ best-score then
27:                    best-score ← sc
28:                    best-sq[ply..] ← move + sq
29:                end if
30:            end for
31:            next-move ← best-sq[ply]
32:            ply ← ply + 1
33:            c-st ← play(c-st, next-move)
34:        end while
35:        return (best-score, best-sq)
36:    end if
37: end function
```

As you can see there is only one FOR loop iterating over the moves in this implementation of the LNMCS, which means the evaluation is incomplete when the algorithm decides whether to prune the first branches or not. This is a minor flaw in this version of the algorithm and it is easily fixed by ulterior versions (along with other shortcomings). Nonetheless, the experiments were made with this "prototype" version of the algorithm.

4 Results

4.1 Lazy Nested Monte Carlo Search

Experimental setup We conducted experiments on the 10 molecules with 48 mers from the benchmark we can find in Holger's [16] and Hsu's [9] work, here on table 1. -E* denotes (the opposite of) the speculated energy of the lowest energy state.

ID	molecule	-E*
1	HPHHPPHHHHPHHHPPHHPPHPH HHPHPHHPPHHPPPHPPPPPPPPHH	32
2	HHHHPHHPHHHHHPPHPPHHPPH PPPPPPHPPHPPPHPPHHPPHHHPH	34
3	PHPHHPHHHHHHPPHPHPPHPHH PHPHPPPHPPHHPPHHPPHPHPPHP	34
4	PHPHHPPHPHHHPPHHPHHPPPH HHHHPPHPHHPHPHPPPPHPPHPHP	33
5	PPHPPPHPHHHHPPHHHHPHHPH HHPPHPHPHPPHPPPPPPHHPHHPH	32
6	HHHPPPHHPHPHHPHHPHHPHPP PPPPPHPHPPHPPPHPPHHHHHHPH	32
7	PHPPPPHPHHHPHPHHHHPHHPH HPPPHPHPPPHHHPPHHPPHHPPPH	32
8	PHHPHHHPHHHHPPHHHPPPPPP HPHHPPHHPHPPPHHPHPHPHHPPP	31
9	PHPHPPPPHPHPHPPHPHHHHHH PPHHHPHPPHPHHPPHPHHHPPPPH	34
10	PHHPPPPPPHHPPPHHHPHPPHP HHPPHPPHPPHHPPHHHHHHHPPHH	33

Table 1: The benchmark's molecules and their speculated lowest energy state

To obtain our results, we used the lazy NMCS with a timeout of 150s, if the algorithm has not found a conformation with the lowest known energy before the end of the timeout then we restart the algorithm until that happens. In our experiments, the playout biased growth gives an immediate gain of 1 to any legal H-H connection, but also a penalty of -0.2 to the H-P connections, this was

made in order to incite the biased growth to keep a maximum of H mers open to future connections, we did not experiment on that variable.

Molecule 4 used a lazy NMCS with a threshold based on the mean of the evaluation playouts with the following parameters:

level	4
#eval playouts	10
pruning ratio	0.97
playout bias	20

The other molecules used a lazy NMCS with a threshold based on the best average from a batch of evaluation playouts with the following parameters:

level	5
#eval playouts	20
pruning ratio	0.9
playout bias	20

Results Different methods of evaluation and pruning can greatly change the performance of the algorithm, and some methods can be ineffective on a set of molecules while being capable on another set.

ID	mean time	interquartile
1	5.5	5
2	12.5	15.5
3	10	14
4	20	25
5	10	10
6	+- 180	-
7	+- 60	-
8	13.5	12
9	+- 120	-
10	7	9

Table 2: Mean time and interquatile (in minutes) of LNMCS on the benchmark molecules to reach a state with the lowest state energy

These results are displayed in minutes and were obtained on a 3.50GHz Intel core i5-6600K CPU, the lowest state energy are the -E* featured in table 1.

Our LNMCS performed very poorly on molecules 6 and 7, we were not able to gather enough data to compute the statistics. This was unexpected since only

molecules 4 and 9 are difficult for PERM (the state of the art chain growth algorithm) to solve according to Grassberger and Hsu's latest paper [9], and molecule 4 posed fewer problems. However, the lazy NMCS could attain the second best energy level very reliably in less than 150s in half of the launches with both molecules.

LNMCS was also able to easily reach the second best level of energy on molecule 9 but could only reach the optimal state every 2 hours approximately, that result was expected since PERM encounters difficulties with that molecule too.

4.2 Comparing LNMCS to Deep Reinforcement Learning in AlphaZero style

Experimental setup Recent results on the HP model were brought to our attention in Deng et al's work [3]. Experiments were realized on 8 other HP strings in table 3 using a neural network based Monte Carlo Search method, in AlphaZero's manner. To the best of our knowledge, this is a first for the HP model.

ID	molecule	-E*
S1	HPHPPHHPHPPHPHHPPHPH	11
S2	HHPPHPPHPPHPPHPPHPPHPPHH	13
S3	PPHPPHHPPPPHHPPPPHHPPPPHH	9
S4	PPPHHPPHHPPPPPHHHHHHHPPHH PPPPHHPPHPP	18
S5	PPHPPHHPPHHPPPPPHHHHHHHHH HPPPPPPHHPPHHPPHPPHHHHH	31
S6	HHPHPHPHPHHHHPHPPPHPPPHPP PPHPPPHPPPHPHHHHPHPHPHPHH	31
S7	PPHHHPHHHHHHHHPPPHHHHHHHH HHPHPPPHHHHHHHHHHHHPPPP HHHHHHPHHPHP	52
S8	HHHHHHHHHHHHPHPHPPHHPPHHP PHPPHHPPHHPPHPPHHPPHHPPHPHP HHHHHHHHHHHH	56

Table 3: The second benchmark's molecules and their speculated lowest energy state

We launched our LNMCS with a threshold based on the best average from a batch of evaluation playouts with the same parameters as with most previous results:

level	5
#eval playouts	20
pruning ratio	0.9
playout bias	20

Results In table 4 we show that LNMCS is able to outperform this recent method as well.

ID	our -E*	LNMCS time to previous -E*	previous -E*
S1	11	30s	11
S2	13	25s	13
S3	9	5s	9
S4	18	0.73s	18
S5	31	103s	31
S6	31	143s	31
S7	54	14s	52
S8	58	135s	56

Table 4: Best scores obtained by LNMCS on the second benchmark molecules

Due to a typo on S6 in [3], molecule S6 was retrieved from [13]. The times displayed in table 4 show approximately how much time is necessary to reach the previous lowest energy state for each molecule.

As you can see in table 4, our LNMCS was able to very quickly get to the lowest know state energy and was even able to find new optimums for molecules S7 and S8. However, despite the LNMCS performing better, our own UCT with biased playouts is inferior to their UCT with the priors approach. We think using priors on the LNMCS could lead to even better results than the ones presented in this paper.

4.3 Other MCS algorithms

We tried to solve The HP model with a variety of different Monte Carlo algorithms. We only applied these algorithms to the first molecule from the benchmark, the results presented in this section are only to give an idea of these algorithms' performances and do not necessarily reflect their potential on the HP model.

Nested Monte Carlo Search The good performances of the NMCS [1] compared to the other algorithms presented in this section is what made us decide to try to improve it for this problem into the LNMCS: the NMCS was able to find the optimal value on the molecule in less than 10mn.

In Figure 2 and Figure 1 we compare performances of the level 5 NMCS and the level 5 LNMCS with a ratio of 0.9 on molecule 1, with both a playout bias of 20 over 20 runs with a 150s timeout.

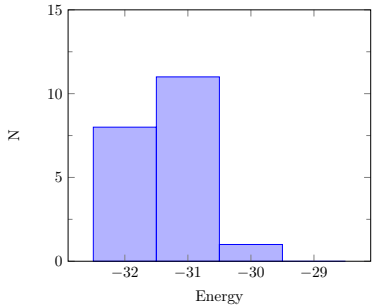


Fig. 1: Energy distribution with the Lazy NMCS

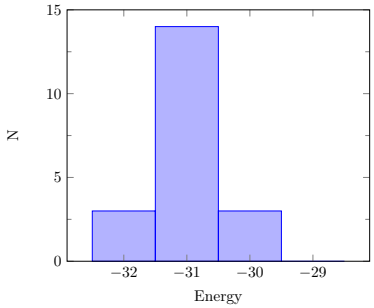


Fig. 2: Energy distribution with the NMCS

As you can see in figures 1 and 2, the LNMCS provides a substantial performance gain over the NMCS on this problem. Lowest energy conformations are way sparser than the second lowest energy conformations, being able to reach them 3 or 4 times as often is a great improvement.

Nested Rollout Policy Adaptation The NRPA [12] is similar to the NMCS, the main difference is that the NRPA learns a policy to decide which move to play during playouts, that policy discovery is interesting for many problems which are too complex to implement a handmade policy (like we did here). On many problems the NRPA outperforms the NMCS, however, in this case it was not able to. The performance of the NRPA and GNRPA [2] are widely dependent on the move representations, here it is the number corresponding to the number of residues already placed and its direction and NRPA and GNRPA with a bias of 20 were not able to reach the optimal energies (-28 or -29 for the GNRPA when -32 is the best known). Other moves representations were tried, using the last few previous moves instead of the number of residues already placed for example, but none were able to provide better performances. Our inability to obtain good results with NRPA does not mean it is impossible to solve this problem with it.

Greedy Best First Search with playouts The Greedy BFS [5] is a simple search algorithm that uses a ranked list of the nodes to open according to their scores given by an evaluation function. Iteratively, the Greedy BFS opens the best node from the list and launches the evaluation function on every child of this node to insert them in the ranked list. Here we evaluate the children with their results with one or multiple playouts. This method converges rapidly to a "good enough" solution (a local minimum), -29 when the best known is -32 on molecule 1, but then improves very little, it is due to a large number of good

scoring states that do not lead to an optimal solution, making the search too exhaustive. Pruning the search tree could improve the results of this method.

Upper Confidence bounds applied to Trees UCT [11] is the most widely used MCTS method, namely in outstanding works like alphazero [15] and AstraZeneca's AiZynthFinder [6]. It iteratively starts from the initial state, go down the search tree following a formula, and launches layouts once it finds an unvisited state, based on the results of each playout, the value used to determine how to go down the search tree are updated. UCT does not work well on the HP model without biased growth (achieving about the same scores as a single biased playout, -18), with biased growth it achieves scores around -28/-29 on molecule 1, like the other non-NMCS algorithms discussed here.

5 Conclusion

In this paper, we proposed a new (prototype) Monte Carlo Search algorithm that has the advantage to be easier to implement than most usual HP model algorithms and is applicable to more problems. It is also shown to be an improvement on the NMCS algorithm for this specific problem and to outperform novel neural network based methods. We have applied LNMCS to other problems, and it has not been outperformed by any other algorithm on any of these problems yet, we also made slight changes to the algorithm towards a final version (hence the use of the term "prototype" here) in papers soon to be published.

In future works, we aim to apply LNMCS to more problems and find ways to improve its performance. We also aim to use a prior neural network to further improve LNMCS performance on the HP model problem.

You can find our codes for the HP model, the LNMCS, and the other algorithms here: https://github.com/RoucairolMilo/HPmodelICCCI

References

1. Cazenave, T.: Nested Monte-Carlo Search. In: Boutilier, C. (ed.) IJCAI. pp. 456–461 (2009)
2. Cazenave, T.: Generalized nested rollout policy adaptation. In: Monte Carlo Search at IJCAI (2020)
3. Deng, H., Yuan, X., Tian, Y., Hu, J.: Neural-augmented two-stage monte carlo tree search with over-sampling for protein folding in hp model. IEEJ Transactions on Electrical and Electronic Engineering **17**(5), 685–694 (2022)
4. Dill, K.A.: Theory for the folding and stability of globular proteins. Biochemistry **24**(6), 1501–1509 (Mar 1985)
5. Doran, J.E., Michie, D.: Experiments with the graph traverser program. Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences **294**(1437), 235–259 (1966)

6. Genheden, S., Thakkar, A., Chadimová, V., Reymond, J.L., Engkvist, O., Bjerrum, E.: AiZynthFinder: a fast, robust and flexible open-source software for retrosynthetic planning. Journal of Cheminformatics **12**(1), 70 (Dec 2020). https://doi.org/10.1186/s13321-020-00472-1, https://jcheminf.biomedcentral.com/articles/10.1186/s13321-020-00472-1
7. Grassberger, P.: Pruned-enriched Rosenbluth method: Simulations of polymers of chain length up to 1 000 000
8. Gros, F., Gilbert, W., Hiatt, H.H., Attardi, G., Spahr, P.F., Watson, J.D.: Molecular and Biological Characterization of Messenger RNA. Cold Spring Harbor Symposia on Quantitative Biology **26**, 111–132 (Jan 1961), publisher: Cold Spring Harbor Laboratory Press
9. Hsu, H.P., Grassberger, P.: A review of Monte Carlo simulations of polymers with PERM. Journal of Statistical Physics **144**(3), 597–637 (Aug 2011)
10. Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S.A.A., Ballard, A.J., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., Back, T., Petersen, S., Reiman, D., Clancy, E., Zielinski, M., Steinegger, M., Pacholska, M., Berghammer, T., Bodenstein, S., Silver, D., Vinyals, O., Senior, A.W., Kavukcuoglu, K., Kohli, P., Hassabis, D.: Highly accurate protein structure prediction with AlphaFold. Nature **596**(7873), 583–589 (Aug 2021)
11. Kocsis, L., Szepesvári, C.: Bandit Based Monte-Carlo Planning. In: Machine Learning: ECML 2006, vol. 4212, pp. 282–293. Springer Berlin Heidelberg, Berlin, Heidelberg (2006), series Title: Lecture Notes in Computer Science
12. Rosin, C.D.: Nested rollout policy adaptation for Monte Carlo Tree Search. In: IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence. pp. 649–654 (2011)
13. Santana, R., Larrañaga, P., Lozano, J.A.: Protein folding in simplified models with estimation of distribution algorithms. IEEE transactions on Evolutionary Computation **12**(4), 418–438 (2008)
14. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of go with deep neural networks and tree search. Nature **529**, 484–489 (2016)
15. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., Hassabis, D.: Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. arXiv:1712.01815 [cs] (Dec 2017), http://arxiv.org/abs/1712.01815, arXiv: 1712.01815
16. Thachuk, C., Shmygelska, A., Hoos, H.H.: A replica exchange Monte Carlo algorithm for protein folding in the HP model. BMC Bioinformatics **8**(1), 342 (Dec 2007)
17. Wüst, T., Landau, D.P.: Optimized Wang-Landau sampling of lattice polymers: Ground state search and folding thermodynamics of HP model proteins. The Journal of Chemical Physics **137**(6), 064903 (Aug 2012)

3.2 Retrosynthesis

I have always had an interest in chemistry, my advisor invited me to a presentation on chemoinformatics at the Pierre-Gilles de Gennes Foundation. There we met two chemists who recommended we look at AiZynthFinder, a retrosynthesis tool, for using our search algorithms on chemoinformatics problems. Retrosynthesis is the task of finding a chain of reactions from available reactants to produce a desired molecule, it is extremely useful in pharmaceutics and is the most "real world" problem in this thesis. Indeed AiZynthfinder is a well made software that is already using a version of Monte Carlo Search similar to UCT.

We substituted our algorithms to their Monte Carlo Search. Both NMCS and GBFS provided improved performances, solving up to 20% more molecules in the long run with NMCS, and immediately outperforming the base algorithm in the short run with GBFS. Because the molecules from the benchmark we used are gradually more complex, this 20% increase in molecules solved is a great improvement, outperforming ASKCOS, the state of the art retrosynthetic tool from the MIT.

However, much remains to be done. For instance, the publicly available one step retrosynthesis neural network (PUCT prior) provided by AstraZeneca is trained on publicly available data. The structure of the neural network is also very simple. Both of these are being worked on and could yield significant improvements.

This is why Ngoc Trinh Hung Nguyen worked on this problem during his internship and managed to improve the neural network's structure. This improvement yielded a 10% increase in molecules solved. With both the NMCS and the improved neural network, the number of molecules solved was further improved to 39, so a 30% improvement, allowing the modified AiZynthfinder to rival or outperform tools using much larger datasets.

The performances could be even further improved by using said datasets to train the new network and as the available list of reactants. But the SOTA has already been outperformed without these so the results are very satisfactory as they are.

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 pretrained 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.

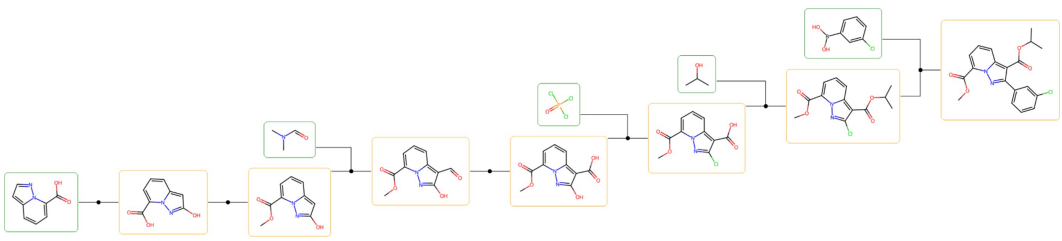


FIGURE 1 Retrosynthetic route for molecule A0.

- *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

```
function MCTS(state)
    if terminal(state) then return score(state)
    end if
    if state is not in the transposition table then
        add state to the transposition table
        return score(state)
    else
        best-score ← −∞
        mean ← (sumEvals(state)+prior(state,m)) / (visits(state)+1)
        for each m in M_state do
            μ ← mean
            if visits(play(state, m)) > 0 then
                μ ← (sumEvals(play(state,m))+prior(state,m)) / (visits(play(state,m))+1)
            end if
            bandit ← μ + c × sqrt((2×(visits(state)+1)) / (visits(play(state,m))+1))
            if bandit > best-score then
                best-score ← bandit
                best-move ← m
            end if
        end for
        res ← MCTS(play(state, best-move))
        update the transposition table
        return res
    end if
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.

```
function NMCS(c-st, lv)
    if lv = 0 then                                          ▷ Playout if lv = 0
        ply ← 0
        seq ← {}
        while not terminal(c-st) do
            move ← heuriChoice(M_c-st)
            c-st ← play(c-st, move)
            seq[ply] ← move
            ply ← ply + 1
        end while
        return (score (c-st), seq)
    else
        best-score ← −∞
        best-sequence ← [ ]
        ply ← 0
        while c-st is not terminal do
            for each move in M_c-st do
                n-st ← play(c-st, move)
                (score, seq) ←NMCS(n-st, level − 1)
                if score ≥ best-score then
                    best-score ← score
                    best-sequence[ply..] ← move + seq
                end if
            end for
            next-move ← best-sequence[ply]
            ply ← ply + 1
            c-st ← play(c-st, next-move)
        end while
        return (best-score, best-sequence)
    end if
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.

```
function val(s1, s2)
    if biggest molecule from s1 is smaller than biggest molecule from s2 then
        return score(s1) + 1
    end if
    return score(s1)
end function
function HeuriChoice(moves)
    weights ← [val(play(st, m)) for m in M_st]
    su ← 0
    rd ← randomRange(0, sum(weights))
    ind ← 0
    for w in weights do
        su ← su + w
        if su ≥ rd then return ind
        end if
        ind ← ind + 1
    end for
    return ind
end function
```

Algorithm 4 The GBFS algorithm.

```
function GBFS(ini-state, max-iter)
    open-states ← [ini-state]
    state ← ini-state
    iter ← 0
    best-state ← ini-state
    best-score ← score(ini-state)
    while not optimal(state) and open-states ≠ [ ] and iter < max-iter do
        iter ← iter + 1
        state ← pop( open-states, 0)
        for each move in M_state do
            new-state ← play(state, move)
            score ← score(new-state)
            insert(open-states, score, new-state)
            if score ≥ best-score then
                best-state ← state
                best-score ← score
            end if
        end for
    end while
    return best-state
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

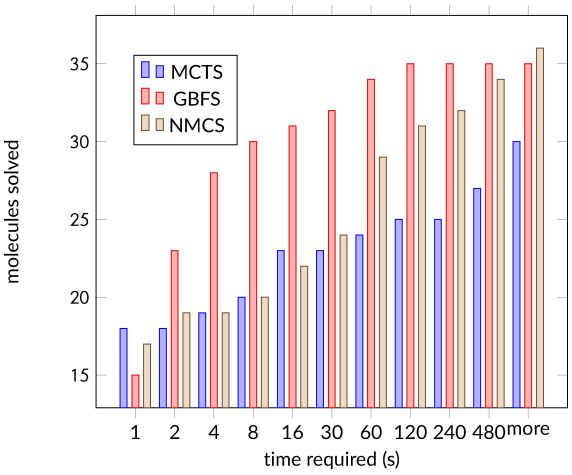


FIGURE 2 Distribution of the numbers of molecules solved with times in seconds.

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.

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.

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

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

Application of Metric Transformation in One-Step Retrosynthesis

Ngoc Trinh Hung Nguyen, Roucairol Milo, Tristan Cazenave

LAMSADE, Université Paris Dauphine - PSL

Abstract

In this article, we investigate the impact of Deep Metric Learning and Transformer architecture on predicting the retrosynthesis of Simplified Molecular Input Line Entry System (SMILES) chemical compounds.

We demonstrate that combining the Attention mechanism with Proxy Anchor Loss is effective for classification tasks due to its strengths in capturing both local and global contexts and differentiating between various classes.

Our approach, which requires no prior chemical knowledge, achieves promising results on the USPTO-FULL dataset, with accuracies of 53.4%, 83.8%, 90.6%, and 97.5% for top-1, top-5, top-10, and top-50 predictions, respectively.

We further validate the practical application of our approach by correctly predicting the retrosynthesis pathways for 63 out of 100 randomly selected compounds from the ChEMBL database and for 39 out of 60 compounds selected by Bayer's chemists and from PubChem.

1 INTRODUCTION

Designing molecules and materials with desired properties is a practical goal of chemistry and materials science. Chemoinformatics involves the use of data and computational methods to investigate and comprehend the connections between molecular structures and their properties, paving the way for the discovery of novel functional molecules.

Chemists prioritize designing synthesis pathways that generate target molecules through a sequence of chemical reactions. A common approach, known as retrosynthetic analysis, involves deconstructing target molecules into precursor molecules and then further deconstructing these precursors until all are available in the stock database for synthesis.

Retrosynthetic planning involves two primary tasks: single-step retrosynthesis prediction and multi-step retrosynthetic planning.

Single-step retrosynthesis is treated as a prediction problem, where the input is a given product molecule and the output is the predicted set of reactant molecules.

Multi-step retrosynthesis planning aims at producing a relizable synthetic route for a specific compound by breaking it down into simpler intermediates or precursors. This process is carried out through retrosynthetic analysis, where the desired compound is iteratively deconstructed until one of the stopping criteria is met: identifying known or purchasable building blocks, reaching a time limit, reaching a maximum tree depth, or fulfilling other predefined criteria ([25]). Multi-step retrosynthesis planning is approached as a search problem, guided by solutions derived from single-step retrosynthesis.

There are two main approaches to the one-step retrosynthesis problem: template-based methods and template-free methods.

Template-based methods mimic chemists' reasoning by using a pool of reaction templates to identify potential reaction centers. The goal is to find the appropriate template that allows the deconstruction of product molecules into their reactant molecules. Template-based methods are a classical approach, where the one-step retrosynthesis problem is considered a classification problem. This approach is reliable since it uses experts' knowledge about chemical reactions and mimics the way chemists work, which makes it easy to use for experts ([22]). This is also the approach for our experiments in this paper.

Template-free methods, on the other hand, are considered sequence-to-sequence learning processes, where the input consists of string-like representations of chemical products (SMILES), and the output corresponds to the SMILES strings of reactants ([22]). The most common models used in this approach are Transformer-based ([33]).

2 METHODS AND DATASETS

2.1 Dataset

2.1.1 USPTO, AiZynthFinder, AiZynthTrain, Reaction Utils and RXNMapper

USPTO-FULL is a dataset extracted from chemical reactions in US patents (1976-Sep 2016) ([19]). It consists of approximately 3.5 million reactions with almost 1.2 million reaction templates.

Several free and open-source software tools are available for downloading and preprocessing the USPTO-FULL reaction dataset. We utilize AiZynthTrain (AZT) ([14]), Reaction Utils ([17]), and RXNMapper ([27]) to prepare and train our one-step retrosynthesis model due to their robustness and comprehensive end-to-end pipelines.

RXNMapper and Reaction Utils are used for downloading, preparing, and performing atom mapping on the USPTO dataset. This is followed by AiZynthTrain, which provides a collection of routines, configurations, and pipelines

for transforming the atom-mapped dataset into molecular fingerprints[1], splitting the data, and training one-step retrosynthesis prediction models. These pipelines also generate human-readable reports, covering all stages from data preprocessing to the training process.

On the other hand, AiZynthFinder (AZF) ([25]) is a free and open-source software for multi-step retrosynthetic planning. Its primary search algorithm is based on Monte Carlo Tree Search (MCTS) ([3]), which recursively breaks down a target molecule into purchasable precursors. We use AiZynthFinder and its MCTS to conduct our experiments on multi-step retrosynthesis tasks. Additionally, we incorporate Nested Monte Carlo Search (NMCS) ([4]), which has already demonstrated promising results in multi-step retrosynthesis ([24]), in order to gain a deeper understanding of the interaction between one-step retrosynthesis models and search algorithms in multi-step retrosynthesis problems.

2.1.2 Dataset

Our primary objective is to enhance the performance of our one-step retrosynthesis model to subsequently improve multi-step retrosynthesis planning with AiZynthFinder ([25]). This objective builds on the work of comparing search algorithms in multi-step retrosynthesis ([24]), including Nested Monte Carlo Search ([4]) and Greedy Best First Search ([10]). To achieve this, we aim to design a robust one-step retrosynthesis architecture that can deliver satisfactory performance within defined time constraints for inference, in order to efficiently support multi-step retrosynthesis planning.

Therefore, we use the USPTO-FULL dataset (as opposed to the USPTO-50k dataset ([26]) used in other studies). This dataset was downloaded and prepared using RXNUtils and RXNMapper, and subsequently filtered with AiZynthTrain. To ensure a fair comparison with the AiZynthTrain base model and its associated experiments, we applied all default filters provided by AiZynthTrain ([14]), including the criterion to retain only templates with more than $N = 3$ example reactions in the USPTO-FULL dataset.

After filtering, the dataset is divided into two subsets: 812,948 samples associated with 42,134 unique reaction templates for the one-step retrosynthesis model, and a "ring-breaker" dataset containing 57,227 ring reactions with 5,225 reaction templates, which is used to train the ring-breaker model ([32]). The one-step retrosynthesis dataset is split into 658,907/97,776/56,265 for training/validation/testing. The ring-breaker dataset is split as 48,800/4,044/4,383 for training/validation/testing.

AiZynthTrain also transformed each molecule into its molecular fingerprint, which serves as the input for the models. The associated labels are the reaction templates that need to be applied to deconstruct the molecules into smaller reactants.

[1] A numerical representation of the molecule, mapping it to a sparse discrete representation space. Molecular fingerprints provide a computationally efficient and consistent way to represent the chemical properties (structural, physicochemical, etc.) of large-scale chemical datasets ([34]).

A key feature of the USPTO-FULL dataset is the large number of reaction templates (or classes) and the significant imbalance between them, with sample counts ranging from 1 for the smallest templates to nearly 8,000 for the largest. This imbalance makes training more difficult, especially for underrepresented classes.

For a clearer assessment with smaller datasets, we also performed experiments on the USPTO-50k dataset ([26]). After AiZynthTrain filtering, we obtained 27,704/4,580/2,376 samples for training/validation/testing, along with a negligible ring-breaker dataset. Like USPTO-FULL, the filtered USPTO-50k is heavily imbalanced, with nearly half the templates having just 1-3 training samples, making it prone to overfitting.

2.2 Methods

2.2.1 Backbone Model

Deep Metric Learning and Proxy Anchor Loss

Despite their extensive use in supervised deep-learning applications, including one-step retrosynthesis problems, cross-entropy-based loss functions are often less effective when there is significant intra-class variance and minimal inter-class variance within the input data distribution.

Deep Metric Learning aims to measure the similarity between data samples by learning a representation function that maps these samples into an embedding space, where samples from the same class are closely grouped together.

The quality of an input's representation largely depends on the loss functions used to train the networks. These loss functions are typically categorized into two classes: pair-based and proxy-based.

Pair-based losses are derived from pairwise distances between data points in the embedding space. A well-known example is Contrastive Loss ([2]), which aims to minimize the distance between pairs of data points with identical class labels and maximize the distance between those with different labels. These losses provide rich data-to-data information by directly comparing data points. However, this approach results in extremely high training complexity and requires a specific arrangement of data to generate both negative and positive pairs.

Proxy-based losses address this issue by introducing proxies as representatives, which are learned as part of the network parameters. Each data point is associated with proxies, typically one per class, enabling the model to leverage data-to-proxy relationships. In essence, a proxy can be thought of as a learnable centroid for each class. The model's task is to adjust the representations of both samples and proxies so that each sample is close to its corresponding proxy, each proxy is close to its respective samples, and optionally, the distance between samples of different classes is maximized. Examples of such losses include Proxy NCA ([23]) and Proxy Anchor Loss ([18]).

We use Proxy Anchor Loss to train our backbone model because it captures both data-to-proxy and data-to-data relationships by taking each proxy as an

anchor and associating it with the entire data in a batch. The loss is given by ([18])

$$\begin{aligned}\mathcal{L}(X) =& \frac{1}{|P^+|} \sum_{p \in P^+} \log \left(1 + \sum_{x \in X_p^+} e^{-\alpha(s(x,p)-\delta)} \right) \\ &+ \frac{1}{|P|} \sum_{p \in P} \log \left(1 + \sum_{x \in X_p^-} e^{\alpha(s(x,p)+\delta)} \right)\end{aligned} \tag{1}$$

In equation 1, $\delta > 0$ is a margin, $\alpha > 0$ is a scaling factor, P indicates the set of all proxies, and P^+ denotes the set of positive proxies of data in the batch. Also, for each proxy p, a batch of embedding vectors X is divided into two sets: X^+, the set of positive embedding vectors of p, and $X^- = X - X^+$ ([18]).

One issue we observed with Proxy Anchor Loss during training is the potential for gradient instability due to the large number of unique reaction templates. Therefore, careful selection of hyperparameters, particularly α, is crucial. We experimented with α values ranging from 1 to 128. With small α values such as 1, 2, and 4, the interaction between samples and proxies was too weak, while larger values such as 64 and 96 led to overly aggressive interactions, causing gradient instability. We found that $\alpha = 32$ yielded optimal results, avoiding gradient issues, consistent with the findings in the Proxy Anchor Loss paper ([18]).

Attention and Positional Embedding

Transformers ([33]) represent the State-of-the-Art in Natural Language Processing due to their ability to capture both local and global context. The Transformer architecture has also achieved notable success in Computer Vision, as demonstrated by the Vision Transformer (ViT) ([11]).

Drawing inspiration from both the original Transformer and the Vision Transformer, we conceptualize a chemical compound as a sentence composed of words. Consequently, we transform the input into vectors of consistent size to apply multi-head self-attention, the mechanism used in Transformer model

We treat a molecule as a sentence, where the position of each 'word' is crucial for the model's effective interpretation. Thus, we incorporate Positional Embedding before applying the attention mechanism.

Metric Transform

We refer to our backbone model, which incorporates Positional Embedding, the Attention mechanism of the Transformer, and Proxy Anchor loss, as the Metric Transform (see Fig 2). The output of the final Attention layer is passed through a Global Max-Pooling 1D layer to capture the most significant information about the input molecule. The Max-Pooling operation reduces the spatial dimensions of the feature maps by selecting the maximum value within a specified spatial window, thereby helping to mitigate overfitting. Additionally, one or more fully connected layers can be added after Max-Pooling to improve the model's ability to distinguish between molecules and facilitate fine-tuning.

This model is then trained using Proxy Anchor Loss to enhance its ability to distinguish between samples from the same class and samples from different classes.

Typically, proxies are initialized and trained with a high learning rate on a pre-trained model, such as ResNet ([15]) or ViT. This approach allows the proxies to quickly adapt and subsequently refine the model parameters to enhance performance. In contrast, our approach involves training the backbone model from scratch. Consequently, we initialize and train the proxies using the same learning rate as the backbone model to encourage the model's parameters to be optimized at a similar pace to the proxies, instead of letting the proxies take the lead as in the original implementations. We also experimented with using a higher learning rate for the proxies and, conversely, a lower learning rate for the proxies. In both cases, the results were less attractive[2] compared to using the same learning rate for both the proxies and the model's parameters.

2.2.2 Subclass Mapping and Fine-Tuning Process

Following a traditional fine-tuning approach, we enhanced our backbone model by adding one or more fully connected layers. This resulted in a 3-4% improvement in performance over the AiZynthTrain base model, demonstrating satisfactory progress.

Unlike moderate-sized classes, which tend to form well-defined clusters during the representation learning process, samples from the major classes exhibit only localized clustering. While our approach improves the model's predictive performance compared to the base model, the inherent bias of deep neural networks toward the major classes restricts the model's ability to generalize effectively to rare templates.

We address this problem by performing "subclass mapping" (see Fig 1) for all major classes using k-means clustering ([20]). Instead of treating a given major class as a single entity, we divide it into smaller subclasses, such that samples within each subclass are closer to one another. This approach has two main benefits:

1. Reduces the imbalance between major and minor classes.

2. Makes it easier for the model to distinguish similarities and differences by using subclass labels instead of whole-class labels.

[2]By "attractive," we refer to the capability of distinguishing samples from minor classes, as the major classes can be handled by a solution we propose below.

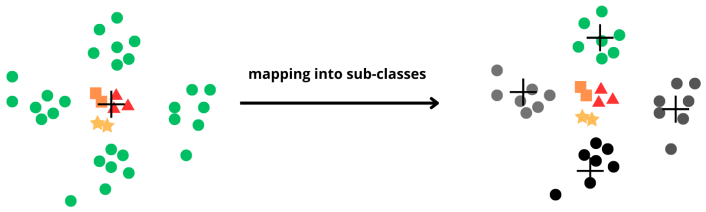


Figure 1: Schematic diagram of the subclass mapping process. Although the green points are locally clustered, their centroid is close to samples from minor classes, which makes distinguishing between classes less effective. We use the subclass mapping to address this issue.

After this step, the number of classes will increase compared to the original dataset. Despite the greater number of classes, this approach helps the model fine-tune more effectively because samples within each subclass are now closer to each other through their local cluster than they were within the original major class. Additionally, the class imbalance will be less severe.

We also define a custom layer that remaps subclasses back to their original classes, enabling reaction template predictions for a given molecule. In the case of a reaction template (class) being split into two or more subclasses, the remapping layer will use predefined criteria to determine the probability of predicting the original class.

Next, we fine-tune our backbone model by adding a fully connected layer to perform the classification task using traditional cross-entropy loss with subclasses and incorporate the remapping layer to remap the subclasses back to their original classes (see Fig 3).

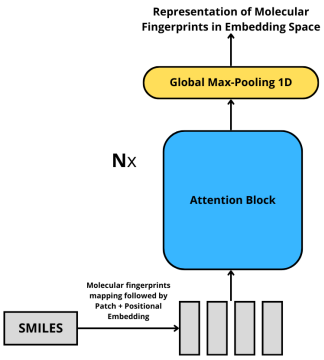


Figure 2: Diagram of the Metric Transform.

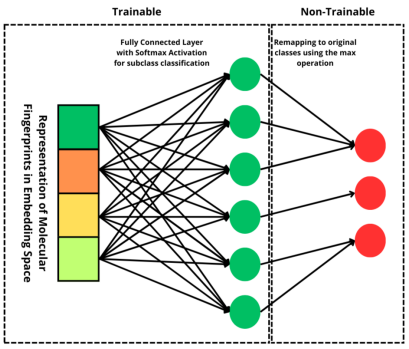


Figure 3: Diagram of the fine-tuning process.

3 RESULTS AND DISCUSSION

3.1 One-step Retro-Synthesis

As described above, our primary objective is to develop a robust model that can be fine-tuned for downstream tasks and applied to other template-based architectures to enhance their performance. Therefore, we trained both our one-step and ring-breaker models using datasets derived from the USPTO-FULL dataset. Each model was trained solely on its training set, with its validation set used for model selection and early stopping criteria..

The entire training process, which included training the backbone model, subclass mapping, and fine-tuning on USPTO-FULL, required only a few hours in total on a 3.30 GHz AMD Ryzen 5 6600H Linux machine with 16 GB of RAM and a NVIDIA GeForce RTX 3050 4 GB GPU.

Comparison between AiZynthTrain base model and Metric Transform

We first compare our architecture with the AiZynthTrain base model for one-step retrosynthesis and the ring-breaker model. It is important to note that the ring-breaker model is trained similarly to the one-step model but on the ring-breaker dataset, a derived dataset that contains only ring reactions.

Model	Top-1 Accuracy	Top-5 Accuracy	Top-10 Accuracy	Top-50 Accuracy
AZT	51.1/52.4	77.3/81.8	83.7/87.9	92/95.1
MetricTransform (Ours)[a]	54.2/51.9	81.4/83.2	87.3/90.4	95/97.3
MetricTransform (Ours)	55/53.4	81.9/83.8	88.1/90.6	95.1/97.5
AZT ring-breaker	70/70.8	90.4/91	93.6/94	97.7/98.2
MetricTransform ring-breaker (Ours)[a]	74.3/74.5	94.9/95.2	97.5/97.9	99.4/99.5
MetricTransform ring-breaker (Ours)	74.3/74.4	95/95.4	97.5/97.7	99.5/99.5

[a] Results without the subclass mapping process.

Table 1: Top-k accuracy of the AiZynthTrain base model and Metric Transform on the USPTO-FULL dataset filtered by AiZynthTrain. Accuracies are reported as the left side of the slash for the validation set, and the right side for the test set.

Comparison of Models Based on Predictive Capacity and Inference Speed

State-of-the-Art models, such as GLN ([8]) and the Augmented Transformer ([31]), utilize the curated USPTO-FULL dataset ([8]). This curated version of the original USPTO-FULL dataset ([19]) was created by removing duplicate reactions and those with incorrect atom mappings, resulting in training/validation/testing sets with 800k/100k/100k samples, respectively.

On the other hand, using the default filtering and splitting of AiZynthTrain, we have dataset of 707,707/101,820/60,648 samples for training/validation/testing. These datasets are split for one-step modeling and ring-breaker modeling. To enable a fair comparison with other models on the USPTO-FULL benchmark, we rescaled our model's performance by applying the following rules:

- Because our models are trained solely on the training set, we rescale accuracies as the average of weighted accuracies from both the one-step model and the ring-breaker model, on both the validation and testing sets.

- Since the ratio between the training set and the validation/testing set is 4.36 (707,707:162,468) instead of the typical 4.0 (696,140:174,035), as used in other papers, this discrepancy implies that some samples should belong to the validation/testing set rather than the training set. Therefore, we consider these 11,567 samples as failed predictions, even though this may not necessarily reflect the actual training and inference process.

- Additionally, we have 25,965 fewer samples[3] compared to the validation/testing sets used in the curated USPTO-FULL dataset ([8]). This shortfall includes samples that belong to rare reaction templates, which only appear once or twice in the entire dataset and have been filtered out by AiZynthTrain. Consequently, we assume failure to predict these samples, similar to the approach used in the Augmented Transformer ([31]).

Furthermore, in the retrosynthesis problem, the primary function of a single-step model is to serve as an environment during the multi-step search process. Therefore, inference time is a crucial metric for evaluating these models, in addition to predictive performance. Faster single-step models can enable more extensive searches within limited time and computational resources ([21]). Despite its importance, inference speed is often overlooked in existing studies. Our work addresses this gap by evaluating models based on their inference time and comparing them with other available models, offering a comprehensive view of inference speed in one-step retrosynthesis.

We present the following comparison of the performances in both predictive capacity and inference speed of models on the USPTO-FULL dataset in Table 2 and 3.

Model	Top-1 Accuracy	Top-10 Accuracy
Retrosim[a]	32.8	56.1
Neuralsym[a]	35.8	60.8
GLN[a]	39.3	63.7
Augmented Transformer[b]	44.4	70.4
AZT[c]	42.7	69.6
MetricTransform (Ours)[c]	45.1	72.7

[a] Results for Retrosim ([6]), Neuralsym ([28]), and GLN as reported by [9].
[b] Results reported by [31] by assuming that Augmented Transformer failed for all 4% of excluded reactions from the curated USPTO-FULL dataset.
[c] Recalculated results on the filtered USPTO-FULL dataset by AiZynthTrain

Table 2: Rescaled Top-k Accuracy of Various Models on the USPTO-FULL Dataset

Model	Inference Speed (sec/sample)
GLN[a]	10^{-1}
Transformer[a]	10^{-1}
MHNreact[a]	10^{-3}
NeuralSym[a]	10^{-3}
AZT[b]	10^{-4}
MetricTransform (Ours)[b]	10^{-4}

[a] Results for GLN, Transformer, MHNreact, and NeuralSym as reported by [29] using Nvidia GPUs (Titan V 12GB, P40 24 GB, V100 16GB, A100 20GB MIG).
[b] Average inference time obtained by evaluating on the whole testing set consisting of 56,265 samples with batch size = 1, using an NVIDIA GeForce RTX 3050 4 GB GPU.

Table 3: Inference speeds of various models, expressed in terms of base-10 exponents

[3]This shortfall is calculated based on the assumption that the total number of samples in the validation/testing sets is 174,035, with 11,567 samples immediately considered as failed predictions because they are actually part of the training set.

Coverage

USPTO datasets have a key characteristic: some reaction rules are absent from the training set. Therefore, the percentage of coverage is often considered the theoretical upper limit for model performance. To better assess predictive capacity, we re-evaluate model performance based on the total coverage of reaction templates, providing a clearer view of each model's learning capabilities.

Model	Coverage	Top-1 Accuracy	Top-5 Accuracy	Top-10 Accuracy	Top-50 Accuracy
GLN[a]	93.3	56.3	81.0	89.7	99.0
GLN[b]	93.3	68.8	91.3	96.5	99.9
LocalRetro[a]	98.1	54.4	87.6	94.2	99.6
LocalRetro[b]	98.1	65.1	94.2	98.2	99.8
LocalRetro[c]	97.0	55.8	81.8	87.0	93.2
AZT[d]	100	52.6	79.6	85.7	93.4
MetricTransform (Ours)[d]	100	55.5	83.2	89.5	96.1

[a] Results on USPTO-50k; reaction class unknown.
[b] Results on USPTO-50k; reaction class known.
[c] Results on USPTO-MIT ([16]).
[d] Results on USPTO-FULL.

Table 4: Rescaled top-k Accuracy of Various Models based on template coverage

USPTO-50k

We conducted quick experiments with the filtered USPTO-50k dataset and compared the AiZynthTrain base model with our approach. Although the dataset is less imbalanced than USPTO-FULL, many templates with only 1-3 samples might complicate the training process and increase the risk of overfitting. Initially, we used Attention layers but found that they caused severe overfitting. Therefore, we replaced them with a fully connected layer in the Metric Transform model and then followed the same process as for USPTO-FULL. Due to the lesser imbalance, models with and without subclass mapping showed no significant performance difference. The results were obtained without knowledge of reaction type.

Model	Top-1 Accuracy	Top-5 Accuracy	Top-10 Accuracy	Top-50 Accuracy
O-GNN	54.1	86.0	92.5	98.3
LocalRetro	53.4	85.9	92.4	97.7
GLN	52.5	75.6	83.7	92.4
AZT[a]	43.5/44.3[b]	70.2/73.9	77.1/81.5	88.7/90.9
MetricTransform (Ours)[a]	48/46.4	75.8/77	82.6/84.9	92.4/94

[a] Results obtained with filtered USPTO-50k
[b] Results on validation/testing set.

Table 5: Top-k accuracy of various models on the USPTO-50k dataset (reaction class unknown)

Discussion of One-Step Retrosynthesis Models

Throughout our experiments, we demonstrate that a combination of the attention mechanism, deep metric learning, and subclass mapping can achieve satisfying predictive performance comparable to state-of-the-art methods, such as

GLN and Augmented Transformer (see Table 2), on the USPTO-FULL dataset while maintaining significantly faster inference times (see Table 3).

Although we achieved good results compared to the AiZynthTrain base model on the filtered USPTO-50k dataset, we observed limitations in our approach with smaller datasets compared to other models such as GLN, LocalRetro, and $\mathcal{O}$-GNN ([35]) (see Table 5).

On the other hand, the experiments (see Table 2 and 4) demonstrate that with larger datasets, state-of-the-art models like GLN and LocalRetro tend to exhibit slightly reduced performance. In contrast, our approaches show improved performance on larger datasets.

We also further investigate subclass mapping and its potential in Section A.1.

3.2 Multi-step Retro-Synthesis

3.2.1 Multi-step Retrosynthesis

We evaluated our model on multi-step retrosynthesis on a 3.30GHz AMD Ryzen 5 6600H, Linux machine with 16 Gb of RAM, NVIDIA GeForce RTX 3050 4GB, using 2 subsets:

- 100 compounds randomly selected from the ChEMBL database ([25]).
- 60 molecules[4], consisting of 20 molecules selected by Bayer's chemists (molecules ID starting with A) and 40 molecules randomly selected from PubChem ([12]), chosen to cover a range of molecular sizes from small to large (molecules ID starting with C).

The first subset was designed to facilitate a comparison between AiZynthFinder and ASKCOS, and since both tools use Monte Carlo Tree Search, our tests on this subset will exclusively use this search algorithm.

The second subset, initially intended to test the predicted difficulty of retrosynthesizing these molecules, features some of the hardest compounds to synthesize, according to Bayer's chemists and Ertl's team. Therefore, we utilize both Monte Carlo Tree Search and Nested Monte Carlo Search to examine the retrosynthetic capacity of the combination of our model with different search algorithms.

We then compare our multi-step retrosynthesis performance with ASKCOS ([5]) and Depth-First Proof-Number Search (DFPN) ([13]). We note the significant differences among them:

- AiZynthFinder and Metric Transform use the USPTO-FULL dataset with the ZINC database ([30]) as stock.
- ASKCOS uses the Reaxys dataset ([1]) with Sigma Aldrich and eMolecules ([7]) as stock .

[4]Dataset available at `https://doi.org/10.5281/zenodo.6511731`

- DFPN uses several public and non-public data sources, consisting of 8,616,239 molecules and 270,605 reaction templates obtained after extensive data cleaning and preprocessing, with chemicals labeled as buyable according to the suppliers for Bayer Research as stock ([13]).

We observe that among these datasets, the USPTO-FULL and ZINC are considerably smaller compared to Reaxys, Sigma Aldrich, and eMolecules. DFPN utilizes a significantly larger dataset, which is ten times the size of USPTO-FULL, along with an internal stock database. However, because DFPN relies on non-public datasets, we are unable to explore their resources further.

Comparison on 100 molecules between ASKCOS, AiZynthFinder and Metric Transform

To ensure a fair comparison, we maintain the original AiZynthFinder setup ([25]). In addition to the number of solved molecules, which serves as the primary criterion in multi-step retrosynthesis, we also present additional statistics to provide a deeper understanding of how the one-step retrosynthesis model impacts solving capacity

Model	Ring Breaker	Iter = 100	Iter = 10,000	Solved
ASKCOS				62
AZF (Original Result)	X	X		55
AZF (Our Test)		X		50
AZF (Our Test)			X	59
AZF (Our Test)	X	X		52
AZF (Our Test)	X		X	60
MetricTransform (Ours)		X		57
MetricTransform (Ours)			X	63
MetricTransform (Ours)	X	X		55
MetricTransform (Ours)	X		X	63

Table 6: Comparison of different models and their performance with various configurations and parameters. The table presents results for models with and without the ring-breaker feature, and across different iteration limits, within a time limit of 120 seconds per molecule.

Model	Average Solution Time	Average Number of Steps	Average Number of Precursors	Solved
ASKCOS	14.3	3.3	3.2	62
AZF (Original Result)	7.1	2.4	2.7	55
AZF (Our Test)	7.7	3.0	3.5	60
MetricTransform (Ours)	6.0	2.8	3.4	63

Table 7: Comparison of retrosynthesis models based on key performance statistics. Results are shown for models with a ring-breaking feature, limited to 120 seconds and 10,000 iterations.

Comparison on 60 molecules between DFPN, AiZynthFinder and Metric Transform

For the AiZynthFinder MCTS experiment, we first set the cutoff number (the maximum number of possible moves returned for a molecule) to 5 with

time limits of 300 and 600 seconds, then increased the cutoff to 50 with limits of 900 and 1200 seconds.

In the NMCS experiment, we also utilize `nmcs1top5` (level 1 NMCS with only the 5 best moves from each state), followed by `nmcs1top50` and `nmcs2top50` ([24]) for more expensive and exhaustive search.

For both experiments, we set the max-transforms (the maximum depth of the search tree) to 7, consistent with the setting used for DFPN.

Model	MCTS[a]	NMCS	MCTS[b]	DFPN
AZF model[c]	31	36		
DFPN model[d]			38	41
Metric Transform (Ours)	33	39		

[a] AiZynthFinder Monte Carlo Tree Search.
[b] Depth-First Proof-Number Search Monte Carlo Tree Search.
[c] Reported results by [24].
[d] Reported results by [13].

Table 8: Comparison of retrosynthesis models using different search algorithms

A comprehensive analysis of the time taken to solve each molecule can be found in Section A.3

Discussion on Multi-Step Retro-Synthesis

We demonstrate the predictive performance of our approach by using different search algorithms on various subsets. Our models yield results comparable to ASKCOS and slightly below DFPN (see Table 6 and 8), despite utilizing a considerably smaller dataset, a reduced stock database, and limited computational resources. Additionally, we reaffirm the potential of NMCS in multi-step retrosynthesis planning, as demonstrated in previous works ([24]).

It is important to note that both the dataset for one-step retrosynthesis modeling and the stock database are crucial in multi-step retrosynthesis planning. A larger dataset can encompass a broader range of reaction templates, while an extensive stock database enhances the flexibility of retrosynthesis planning, resulting in significantly improved outcomes ([25]).

The ring-breaker model yields mixed results, which are discussed in Section A.2.

4 CONCLUSION

The combination of deep metric learning, attention mechanisms, and subclass mapping resulted in a robust architecture that addresses the inherent imbalance in chemical reaction datasets. This approach demonstrates strong performance across both common and rare reaction types, delivering satisfactory results in both one-step and multi-step retrosynthesis. Notably, this was achieved with minimal training time, limited resources, and without modifications to the default AiZynthTrain settings.

Our approach, based solely on Machine Learning and Deep Learning principles, treats the retrosynthesis problem as a classification challenge. We believe

that this method can also be adapted and applied to other template-based approaches to enhance their performances.

5 FUTURE WORKS

We recognize that our exploration of this field is just beginning. Notably, we have not yet transformed reaction templates to reduce the number of templates requiring classification—a crucial step demonstrated in other top-performing studies to enhance performance.

Future work will involve training a model on larger datasets such as Reaxys, and integrating USPTO-FULL with synthetic reaction datasets. Additionally, we plan to delve deeper into chemical and molecular structures to transform the dataset, including optimizing the structure of reaction templates to reduce the actual number of templates necessary for classification. We also aim to adapt our approach to current State-of-the-Art models like: LocalRetro, GLN to evaluate whether our approach can improve their performances.

Acknowledgments

We would like to acknowledge the contributions of the authors of AiZynthFinder, AiZynthTrain, RXNMapper, and Reaction Utils for making these valuable tools open-source, which greatly facilitated solving synthesis problems.

References

[1] *Reaxys*. Elsevier, 2019. Copyright © 2019 Elsevier Limited except certain content provided by third parties. Reaxys is a trademark of Elsevier.

[2] J. Bromley, J. Bentz, L. Bottou, I. Guyon, Y. Lecun, C. Moore, E. Sackinger, and R. Shah, *Signature verification using a "siamese" time delay neural network*, International Journal of Pattern Recognition and Artificial Intelligence, 7 (1993), p. 25.

[3] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez Liebana, S. Samothrakis, and S. Colton, *A survey of monte carlo tree search methods*, IEEE Transactions on Computational Intelligence and AI in Games, 4:1 (2012), pp. 1–43.

[4] T. Cazenave, *Nested monte-carlo search.*, in IJCAI International Joint Conference on Artificial Intelligence, 2009, pp. 456–461.

[5] C. W. Coley, R. Barzilay, T. S. Jaakkola, W. H. Green, and K. F. Jensen, *Prediction of organic reaction outcomes using machine learning*, ACS Central Science, 3 (2017), pp. 434–443.

[6] C. W. Coley, L. Rogers, W. H. Green, and K. F. Jensen, *Computer-assisted retrosynthesis based on molecular similarity*, ACS Central Science, 3 (2017), pp. 1237–1245. PMID: 29296663.

[7] C. W. Coley, D. A. Thomas, J. A. M. Lummiss, J. N. Jaworski, C. P. Breen, V. Schultz, T. Hart, J. S. Fishman, L. Rogers, H. Gao, R. W. Hicklin, P. P. Plehiers, J. Byington, J. S. Piotti, W. H. Green, A. J. Hart, T. F. Jamison, and K. F. Jensen, *A robotic platform for flow synthesis of organic compounds informed by ai planning*, Science, 365 (2019), p. eaax1566.

[8] H. Dai, C. Li, C. Coley, B. Dai, and L. Song, *Retrosynthesis prediction with conditional graph logic network*, in Advances in Neural Information Processing Systems, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds., vol. 32, Curran Associates, Inc., 2019.

[9] H. Dai, C. Li, C. W. Coley, B. Dai, and L. Song, *Retrosynthesis prediction with conditional graph logic network*, ArXiv, abs/2001.01408 (2020).

[10] J. E. Doran and D. Michie, *Experiments with the graph traverser program*, Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences, 294 (1966), pp. 235–259.

[11] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, *An image is worth 16x16 words: Transformers for image recognition at scale*, in International Conference on Learning Representations, 2021.

[12] P. Ertl and A. Schuffenhauer, *Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions*, Journal of Cheminformatics, 1 (2009), p. 8.

[13] C. Franz, G. Mogk, T. Mrziglod, and 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, IJCAI-22, L. D. Raedt, ed., International Joint Conferences on Artificial Intelligence Organization, 7 2022, pp. 4747–4753. Main Track.

[14] S. Genheden, P.-O. Norrby, and O. Engkvist, *Aizynthtrain: robust, reproducible, and extensible pipelines for training synthesis prediction models*. ChemRxiv, 2022. This content is a preprint and has not been peer-reviewed.

[15] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[16] W. Jin, C. Coley, R. Barzilay, and T. Jaakkola, *Predicting organic reaction outcomes with weisfeiler-lehman network*, in Advances in Neural Information Processing Systems, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds., vol. 30, Curran Associates, Inc., 2017.

[17] C. Kannas, A. Thakkar, E. Bjerrum, and S. Genheden, *rxnutils – a cheminformatics python library for manipulating chemical reaction data.* ChemRxiv, 2022. This content is a preprint and has not been peer-reviewed.

[18] S. Kim, D. Kim, M. Cho, and S. Kwak, *Proxy anchor loss for deep metric learning*, 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), (2020), pp. 3235–3244.

[19] D. Lowe, *Chemical reactions from us patents (1976-sep2016).* `https://doi.org/10.6084/m9.figshare.5104873.v1`, 2017. Figshare dataset.

[20] J. Macqueen, *Some methods for classification and analysis of multivariate observations*, in Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability/University of California Press, 1967.

[21] K. Maziarz, A. Tripp, G. Liu, M. Stanley, S. Xie, P. Gaiński, P. Seidl, and M. Segler, *Re-evaluating retrosynthesis algorithms with syntheseus*, in NeurIPS 2023 AI for Science Workshop, 2023.

[22] Z. Meng, P. Zhao, Y. Yu, and I. King, *A unified view of deep learning for reaction and retrosynthesis prediction: Current status and future challenges*, in Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23, E. Elkind, ed., International Joint Conferences on Artificial Intelligence Organization, 8 2023, pp. 6723–6731. Survey Track.

[23] Y. Movshovitz-Attias, A. Toshev, T. Leung, S. Ioffe, and S. Singh, *No fuss distance metric learning using proxies*, 2017 IEEE International Conference on Computer Vision (ICCV), (2017), pp. 360–368.

[24] M. Roucairol and T. Cazenave, *Comparing search algorithms on the retro-synthesis problem*, Molecular Informatics, 43 (2024).

[25] G. Samuel, T. Amol, C. Veronika, J.-L. Reymond, O. Engkvist, and E. Bjerrum, *Aizynthfinder: a fast, robust and flexible open-source software for retrosynthetic planning*, Journal of Cheminformatics, (2020).

[26] N. Schneider, N. Stiefl, and G. Landrum, *What's what: The (nearly) definitive guide to reaction role assignment*, Journal of Chemical Information and Modeling, 56 (2016).

[27] P. Schwaller, B. Hoover, J.-L. Reymond, H. Strobelt, and T. Laino, *Extraction of organic chemistry grammar from unsupervised learning of chemical reactions*, Science Advances, 7 (2021), p. eabe4166.

[28] M. H. S. Segler and M. P. Waller, *Neural-symbolic machine learning for retrosynthesis and reaction prediction*, Chemistry - A European Journal, 23 (2017), pp. 5966–5971. Epub 2017 Feb 22.

[29] P. Seidl, P. Renz, N. Dyubankova, P. Neves, J. Verhoeven, J. K. Wegner, M. Segler, S. Hochreiter, and G. Klambauer, *Improving few- and zero-shot reaction template prediction using modern hopfield networks*, Journal of Chemical Information and Modeling, 62 (2022), pp. 2111–2120.

[30] T. Sterling and J. J. Irwin, *Zinc 15 – ligand discovery for everyone*, Journal of Chemical Information and Modeling, 55 (2015), pp. 2324–2337. PMID: 26479676.

[31] I. V. Tetko, P. Karpov, R. V. Deursen, and G. Godin, *State-of-the-art augmented nlp transformer models for direct and single-step retrosynthesis*, Nature Communications, 11 (2020), p. 5575.

[32] A. Thakkar, N. Selmi, J.-L. Reymond, O. Engkvist, and E. J. Bjerrum, *"ring breaker": Neural network driven synthesis prediction of the ring system chemical space*, Journal of Medicinal Chemistry, 63 (2020), pp. 8791–8808. PMID: 32352286.

[33] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, *Attention is all you need*, Advances in neural information processing systems, 30 (2017).

[34] J. Yang, Y. Cai, K. Zhao, H. Xie, and X. Chen, *Concepts and applications of chemical fingerprint for hit and lead screening*, Drug Discovery Today, 27 (2022), p. 103356.

[35] J. Zhu, K. Wu, B. Wang, Y. Xia, S. Xie, Q. Meng, L. Wu, T. Qin, W. Zhou, H. Li, and T.-Y. Liu, *$\mathcal{O}$-GNN: incorporating ring priors into molecular modeling*, in The Eleventh International Conference on Learning Representations, 2023.

A Appendix

A.1 Subclass mapping

The number of clusters per class is an important hyperparameter that needs to be fine-tuned because a small number of clusters per class will lead to a small variance (which might cause underfitting), while a large number will provide a small bias (but might cause overfitting).

In our experiments, we set the number of clusters by $\lceil \frac{n}{k} \rceil$, where n is the number of samples in a given class, and k is a constant ranging from 50 to 500. We observed the results after the fine-tuning process: a small number of clusters (i.e., a large k) is usually better on the validation and test sets. On the other

hand, a large number of clusters (i.e., a small k) can give better results on the training set but slightly worse results on the validation and test sets.

We also explored whether using subclass mapping before training the backbone model could be beneficial. This approach is intriguing because it could potentially mitigate the impact of dataset imbalance and improve the embedding process of the backbone model.

However, it is important to note that the input consists solely of molecular fingerprints, which are numerical representations of molecules. These fingerprints may not provide significant information for the subclass mapping process. Our test of this approach resulted in poorer performance compared to our initial method. This notably worse result highlights the importance of a well-defined projection of our Metric Transform model to the embedding space and underscores the need for an effective projection to enhance the utility of the subclass mapping.

To further explore the utility of subclass mapping, we conducted additional experiments:

1. Set the limit of samples for subclasses.

2. Fine-tune the backbone model on the subclassed dataset.

In experiment 1, we observed a slight improvement, making it easier to achieve the results shown in Table 1.

In experiment 2, the final results significantly improved, with performance gains ranging from 0.1 to 0.5 for top-1 to top-50 accuracy.

The reason behind these improvements is that, after subclass mapping, samples from the same class may belong to different subclasses while still sharing characteristics from the original class. Fine-tuning the backbone model on the new subclassed dataset allows for better separation of these subclasses.

Please note that the results on one-step and multi-step retrosynthesis presented above were obtained without incorporating this new exploration.

A.2 Ring-Breaker Model

The idea behind the ring-breaker model is to help the model break ring systems in a given molecule, thereby making it easier to decompose that molecule into smaller and purchasable precursors ([32]).

The mixed results obtained by incorporating the ring-breaker model into multi-step retrosynthesis (see Table 6) can be explained by the following elements:

1. Search algorithms need to allocate separate time for evaluating both the one-step model and the ring-breaker model, which reduces effectiveness when the search time is limited.

2. The ring reaction dataset is not large enough to effectively cover ring systems.

These two points lead to an ineffective situation where the one-step model is already sufficient for solving small and medium molecules, but large molecules remain unsolvable (see Table 6, 8 and 9). Extensive research is needed to effectively incorporate ring-breaker models into multi-step retrosynthesis planning.

A.3 Analysis of the subset of 60 molecules of Bayer's chemists and PubChem

We provide a comprehensive analysis of the time required to solve each molecule in the subset of 60 molecules sourced from Bayer's chemists and PubChem. We note that solving time is highly dependent on computational resources, particularly for more challenging molecules. Therefore, this analysis aims to highlight the differences in performance between MCTS and NMCS.

ID	**Time (s)**	**ID**	**Time (s)**	**ID**	**Time (s)**	**ID**	**Time (s)**
A0	73.1/274.3	A1	0.02/1.25	A2	0.06/21.7	A3	147.9/416.1
A4	27.4/213.1	A5	558.6/252.5	A6	X/1035	A7	131.7/165.4
A8	X/1404	A9	6.6/13.5	A10	X/1420	A11	0.06/4.5
A12	98.6/184	A13	X/1539	A14	0.06/14.8	A15	0.9/41.6
A16	0.3/11.7	A17	0.01/33.6	A18	5.2/116.6	A19	532.6/7.3
C1	0.02/1.3	C2	X/X	C3	0.03/3.4	C4	X/X
C5	0.02/1.3	C6	X/X	C7	X/X	C8	0.03/2.9
C9	X/X	C10	X/X	C11	X/X	C12	X/X
C13	0.04/2.5	C14	0.01/32.4	C15	X/X	C16	X/X
C17	X/X	C18	X/X	C19	22/46.6	C20	10.8/1514.6
C21	0.09/2.7	C22	X/647.9	C23	0.03/1.6	C24	X/X
C25	0.01/1.3	C26	0.02/1.27	C27	X/X	C28	X/X
C29	X/X	C30	X/X	C31	0.01/32.2	C32	X/X
C33	X/X	C34	0.03/2.9	C35	X/2217	C36	1.7/21.1
C37	13.3/81.6	C38	X/X	C39	X/X	C40	0.01/33.9

Table 9: Analysis of the time required (in seconds) to solve each molecule in the subset of 60 molecules from Bayer's chemists and PubChem using Metric-Transform. The values to the left of the slash represent times from Monte Carlo Tree Search, while those to the right correspond to times from Nested Monte Carlo Search. 'X' indicates non-solvable molecules within the time limit of 3600 seconds.

Our analysis indicates that, using the same one-step retrosynthesis model, NMCS generally demonstrates superior performance compared to MCTS. Every molecule successfully solved by MCTS was also addressed by NMCS, although NMCS required more time. One potential strategy is to use MCTS to solve the "easier" molecules first, followed by NMCS for the more challenging ones. This approach would allow us to benefit from the speed of MCTS while leveraging the higher solving performance of NMCS.

We also highlight the challenges of multi-step retrosynthesis, particularly with large molecules such as C2 and C39. These molecules contain complex ring systems that remain unsolvable in our experiments. One potential strategy is to employ a ring-breaker model to simplify the retrosynthesis planning by breaking down these ring systems. However, as discussed in Section A.2, the USPTO-FULL ring breaker model has drawbacks that affect the solving capacity of our model and search algorithms for these large molecules.

Additionally, we encounter difficulties with some medium-sized molecules, such as C6 and C7, which also remain unsolvable. This issue is likely due to the lack of specific templates needed to break down these molecules effectively.

Interestingly, we found that C22 and C38, which remain unsolved using MCTS, have precursors that are not in stock—Br for C22 and O=C1OC(=O)C2C3C=CC(C3)C12 for C38—but these precursors can be easily purchased or found on eMolecules. This highlights the importance of having a large dataset and a detailed stock database to enhance the ability to solve multi-step retrosynthesis problems.

3.3 De Novo Molecule Design

With retrosynthesis and AstraZeneca's AiZynthFinder we have a way to assess if molecules are synthesizable. The next step was to automatize the generation process using our algorithms too. De Novo Molecule Design is another vast subject of research with hundreds of papers. Here we decided to search the molecule space with SMILES (string) representation, similar to an atom-by-atom model.

Despite the large amount of papers, most software use a data-only approach with generative neural networks, or use Monte Carlo Search with neural networks, but do not check the validity of each move.

The particularity of our model is that it cannot produce invalid molecules. The search model is encoded in a way that prevents any failure to produce a valid molecule.

In addition, we used AiZynthFinder to assess the synthesizability of the produced molecules. This resulted in a 32% synthesizability rate wich is surprisingly high given that the score function only maximized the drug-likeness and nothing else. The synthesizability of the produced molecules is a side product but is in line with the performances of AiZynthFinder on the FDA database.

Last, this approach is data-light, ngrams were trained over the FDA-approved drug database containing less than 2000 molecules, and provided better results and synthesizability over the previous state-of-the-art neural networks, as well as being 500 times faster. In addition, molecules generated have high novelty and other metrics show that it is not a regurgitation of the training data set.

This project was able to produce valid, unique, novel, and synthesizable, with an extremely small training set and extremely fast generation time. It is to the best of our knowledge never seen before.

The molecules were only made to be drug-like, the next step is to generate molecules that cure a specific disease by training a neural network on a binding simulation to act as a score function.

DrugSynthMC: an atom based generation of drug-like molecules with Monte Carlo Search

Milo Roucairol,[†] Tristan Cazenave,[†] Olivier E. Pardo,[‡] Filippo Prischi,[‡] and Alexios Georgiou

[†]*LAMSADE, Université Paris-Dauphine, Paris, Pl. du Maréchal de Lattre de Tassigny, 75016 Paris, France*

[‡]*Division of Cancer, Department of Surgery and Cancer, Imperial College, Du Cane Road, London W12 0NN, UK*

E-mail:

Abstract

The design or screening of molecules with desired pharmacokinetic and selectivity profiles is an important focus in translational biology. This paper presents DrugSynthMC, a new method for automated and computer-assisted de novo molecule design using an atom-based research model that builds molecules as SMILES, character by character.

Combined with Monte Carlo Search, we show that DrugSynthMC is able to produce valid and synthesizable drug-like molecules that enforce a set of Lipinski rules in fractions of seconds. This ease of drug-like molecule generation can then be combined with score functions aimed at different goals to design useful molecules. Our approach can function with or without an underlying neural network and is thus easily explainable and versatile. Combined with digital screening pipelines, DrugSynthMC is expected to enable the functional assessment of large compound libraries covering an extensive novel chemical space, overcoming the limitations of existing drug collections.

Introduction

Since the 1980s, in silico approaches have been extensively and routinely used in drug discovery and have transformed the medicinal chemistry field,[1] with the expectation to do so even more in the future. The need for rapid response, highlighted by the emergence of resistant bacteria and, among others, the COVID-19 pandemic, has fuelled the development of novel computational tools for drug design and screening.[2] *In silico* virtual-library screening (VS) is usually the first critical step in structure-based drug discovery, where the algorithm aims to predict the best matching binding mode of a ligand to a receptor.[3] Despite the many attempts to improve accuracy of VS methods,[4,5] the relatively limited chemical diversity of compounds in libraries reduces the ability of structure-based VS to identify hits and leads.[6,7] Indeed, it has been estimated that only a small portion (10^6-10^7) of the 10^{63} drug-like molecules predicted to be synthetically accessible has been explored.[8]

Several studies have shown that screening larger libraries that expand the accessible molecules by several orders of magnitude ($\sim 10^{11}$) improves the rate of true high affinity (nM-pM) binders.[9–12] To further expand the chemical space within virtual libraries, generative models based on Deep Learning (DL) methodologies have been used to produce molecules with desired chemical features able to bind specifically macromolecules of interest (extensively reviewed in[13–15]).

Recurrent neural networks (RNNs) were among the first DL methods to be developed to generate SMILES, a line notation that describes the structure of a molecule[16](Weininger 2002). However, RNNs tend to suffer from exposure bias, and a diverse range of alternative approaches that differ in the training procedure and model architecture have been proposed. These include Variational AutoEncoders (VAEs),[17] Generative Adversarial Networks (GANs),[18] and graph-based generators.[19] Nevertheless, even these alternative approaches have limitations and, for example, it has been reported that VAEs generated SMILES often cannot be translated into interpretable chemical structures.[20] Furthermore, a key requirement of generative models is that designed molecules must be synthesizable. A wide range of

different approaches has been used to predict the synthetic feasibility of molecules, including using scores based on structure complexity and similarity to evaluate synthesizability,[21] or integrating computer-aided synthesis planning (CASP) tools as part of the design process.[22,23] However, as Bilodeau et al.[24] recently highlighted, approaches that embed CASP tools automatically inherit CASP limitations, thus reducing the chemical diversity of compounds in libraries.

In this paper, we use Monte Carlo Search (MCS) algorithms in conjunction with DL and statistical-based priors to generate thousands of interpretable chemical structures and novel drug-like molecules per second. DrugSynthMC (Drug Synthetise using Monte Carlo) relies on an algorithm never previously used in chemistry/medicinal chemistry, differing from prior efforts in that it rapidly produces valid molecules, while being explainable and, importantly, requiring no training. The algorithm does not enforce synthesizability or rely on synthesizability metrics during SMILES generation. However, we analyzed the synthesizability of generated compounds using an open-source retrosynthesis analysis tool, AiZynthFinder.[25] We show that our method generates drug-like libraries with a high proportion of predicted-to-be synthesizable compounds and efficiently expands the chemical space within the libraries. Finally, DrugSynthMC is highly flexible and, in the future, could be easily tuned using multiple parameters to tailor for a wide range of different chemical goals and/or create customized libraries of compounds for specific targets.

DrugSynthMC

Search model

The search model consists of a set of instructions defining what available moves (character addition) can be applied to unfinished SMILES. All SMILES start empty, and only one atom can be added at that stage. Once atoms are added, cycles and subtrees can be initiated. This search model ensures that initiated SMILES can be completed from any point of the

search into a valid final SMILE, with generation being heavily restricted by the below rules.

The central rule is to respect the total number of bonds each atom can protract. To do so, the number of available bonds of the last added atom in the current subtree is stored. This number is checked to compute possible legal moves and decreased, when adding a character corresponding to a new atom in a different subtree level, according to the number of bonds used to connect the new atom to the previous one.

To maximise the chance of generating valid molecules with drug-like properties, we scanned the FDA-approved drug bank from ZINC[?] to obtain frequency information on types of atoms and bonds involved. This information was stored in frequency matrices, allowing to label as illegal moves by the search model bonds that were never or very rarely encountered (less than 1/10000 of the bonds for each atom involved). It also enabled us to focus the generator on using the most commonly encountered atoms only: C, O, N, F, S, and Cl. Hence, Br and P atoms were left out due to their relative rarity, but can easily be reintroduced in the search model, along with inorganic atoms.

Last, we added shortcuts for the different moieties involving sulfur. A sulfur atom can act as a 2-bond atom, a 4-bond atom in sulfinyl, and a 6-bond atom in sulfonyl. Rather than learning the entire functional groups and edge possibilities through the prior, we decided to pre-process the SMILES prior to training, turning the trifluoromethyl (W), sulfinyl (M), and sulfonyl (U) residues into their own symbols used in building SMILES.

Operating principle

To give an accurate list of possible characters to append to an incomplete SMILE, the search model keeps track of several parameters concerning the SMILES at any step.

(1) The depths of the nested subtrees: The subtrees are expressed in SMILES language using the "(" symbol for opening, and the ")" symbol for closure. Termination of SMILES is not allowed unless they are back to the root tree, meaning that all open parentheses must be closed. Closing a subtree when no parenthesis is open is also forbidden.

(2) Covalence bounds counts: Each subtree contains an active atom, which is the last one added at that level. It is the atom to which other atoms, cycles, and subtrees are then added. The search model keeps track of the number of available covalent bounds on the last atom of each subtree until the latter is closed by a ")". Moves that exceed the number of covalent bounds available are not allowed.

(3) Cycle nesting: Nested cycles that share more than one bond are uncommon in drug-like molecules. Thus only the most recent cycle is allowed to be terminated.

(4) Cycle length: More rules, not inherent to the SMILES grammar, were added to improve the drug-likeness and stability of the molecules. These included not allowing cycles smaller than 5 and bigger than 7 atoms to be generated. Indeed, while these cycle sizes exist in drug-like molecules, size 4 cycles are usually hard to synthesize and unstable, while cycles longer than 7 atoms are rare.

Additionally, to avoid unnecessarily long playouts and molecules, a boolean flag called "finish_ASAP" is added. It is set to true once a certain number of characters is met and disallows certain moves, such as opening a new subtree or cycle, with some exceptions (ie. finishing an already open subtree).

This complicated set of rules is necessary to prevent the search from cornering itself due to cycles or using all the covalent bounds available on a particular level. However, this results in a rather lengthy function enumerating the legal moves from an incomplete SMILES (about 100 lines long).

Playouts

Before explaining the Monte Carlo Tree Search and other algorithms used in this study, we believe that it is important to give a clear explanation of the term "playout". In Monte Carlo Search, a playout is a computationally cheap unfolding of actions from a starting search space state. Moves are selected and played until the resulting new search is terminal or no move is available. The terminal state is then evaluated (here using Lipinski rules, see

corresponding section below) and returned to be used by the algorithm calling the playout.

Playouts are a core element of most Monte Carlo Search algorithms and the move selection process differentiates algorithms. Many algorithms use random playouts, usually when tackling NP-hard problems where expert knowledge can't help. The Prior Upper Confidence bounds applied to Trees (PUCT[26]) used in DeepMind's AlphaGo[27] employs a neural network to recommend moves to play in a game of Go. The Nested Rollout Policy Adaptation (NRPA[28]) learns a reinforcement learning policy on the fly to select the moves. Some other applications can evaluate non-terminal states and use greedy playouts (ARTICLE PUBLISHED SOON). In fact, the choice of playout mode can be more important to the success of a MCS than the algorithm used (ARTICLE PUBLISHED SOON). In this study, we used the Sampling method to act as a baseline to compare our algorithms. This method consists of independent playouts from the start state and ends once a molecule reaches the best possible score.

Guided playouts: ngrams

Ngrams are short subsequences derived from larger sequences of characters. They were one of the first machine learning approaches, mostly used in Natural Language Processing (NLP). Through the use of grams, it is possible to compute the statistics of every sequence of characters in a learning corpus to predict the next character in a Markovian process.

Here, ngrams were generated through extracting every sequence of characters from the FDA-approved compound database and were associated conditional probabilities used to guide the playout of an incomplete SMILES given its last characters at any step. For instance, if the learning corpus only contained "COCC" and "COCO", the entry for "COC" would report $P(C|COC) = 0.5$ and $P(O|COC) = 0.5$. The ngrams were only used to value moves and act as a prior, following the rules of the search model. Hence, whenever the ngram valued a move that was forbidden by the model, the move was discarded. The ngrams also used the cycle length computed from the FDA-approved compound database to guaranty the

right proportion of cycles of each length in the final molecules. Storing information about the cycles' length within the ngrams themselves would be possible but expected to be more error-prone. Therefore, we instead decided to use a probability to end a cycle at a certain length. These probabilities are 0.228 for a cycle of length 5, 0.751 for a cycle of length 6, and 0.02 for a cycle of length 7. Smaller and longer cycles were omitted for reasons mentioned above.

Guided playouts: neural network

As previously explored,[29] neural networks can be trained to give conditional probabilities of the next character given all of the SMILES. The neural network is then repeated on the newly extended SMILES until the SMILES is terminal, similar to recent Large Language Models (LLM). This method strays from the usual playouts as it is not limited by the rules of the search model, but is prone to produce invalid smiles.[29]

The same neural network can also be used as a prior, following the rules of the search model just like in the case of ngrams. This method has two advantages over approaches using ngrams: (1) taking the entire context of the SMILES into account, and (2) generalization (as ngrams require an exact precursor). However, ngrams are faster, more easily explainable, and require no training.

The neural network used in the present study is the same as presented in,[29] and was trained using the same dataset. It was only slightly modified to accommodate for the short-cuts with no apparent repercussions on the results obtained.

Monte Carlo Search

Monte Carlo Search (MCS) encompasses a wide range of search algorithms. These differ from regular search algorithms in that they are not deterministic and use randomness to explore search spaces too large for regular algorithms and learn guiding policies. The first Monte Carlo algorithm was introduced by Nicholas Metropolis in 1949 with Markov Chain Monte

Carlo.[30] In 2006, Kocsis introduced a variant of Monte Carlo Tree Search (MCTS) called Upper Confidence bounds applied to Trees (UCT) or Bandit based Monte-Carlo Planning.[31] It is now the most widely used MCTS and MCS algorithm in the literature, often in the form of Prior UCT (PUCT). MCS algorithms have since seen a wide range of uses: protein folding,[32] inverse RNA folding,[33–35] retrosynthesis,[25] multi-agent systems,[36] game playing,[37] network optimization,[38] and conjecture refutation.[39] In 2016, Deepmind's Alphago famously won four games of Go out of five against one of the best players in the world using a variant of UCT:PUCT, combining MCTS and neural networks. it was the first time in history a computer achieved superhuman performance on Go. Indeed, Go search space was too large to use the same algorithms as in Chess (i.e. Deep Blue). In short, Monte Carlo Search algorithms are general and have been applied to many different combinatorial optimization problems that have a state space too large to be completely explored.

Upper Confidence bounds applied to Trees

UCT is the most commonly used MCTS algorithm. It is a bandit-based reinforcement learning algorithm similar to Q-learning. The algorithm learns a policy and selects to go down the tree in order to balance exploitation and exploration.

Like all MCTS, UCT is comprised of 4 phases:

1. Selection: Progress in the selection tree according to the policy.
2. Expansion: Once a state that has not been explored yet is encountered, it is added to the tree.
3. Evaluation: A playout (or another fast algorithm) is used to evaluate the quality of the new state.
4. Backpropagation: The result of the evaluation is used to update all the parent states visited during the Selection step.

These four steps are repeated indefinitely, starting from the initial state each time, much like in Q-learning. What differs between the various MCTS algorithms is the formula of the policy.

UCT uses the following formula to evaluate a child state S to select:

$UCT_S = X_S + C * \sqrt{\frac{\ln V}{V_S}}$

Where X_S is the average score of state S, C is the exploration/exploitation constant (usually 1), V is the number of visits of the current state, and V_S is the number of child state S visits.

PUCT is a generalization of UCT. It uses a prior to guide not only the playouts but also the selection process, allowing to speed up the latter with knowledge from outside this execution. PUCT uses a different selection formula:

$UCT_S = X_S + C * P_S * \frac{\sqrt{V}}{1+V_S}$

With P_S the value given by the prior for state S.

Nested Monte Carlo Search

Nested Monte Carlo Search (NMCS) is a different type of MCS algorithm. It uses lower-level NMCS on each available move of the current state to explore the search tree and register the sequences of actions leading to the best scores. Once the lower-level NMCS returns their best routes to the higher level, it executes the next move of the best route and calls new lower-level NMCS on the resulting state. The lowest level NMCS is (usually) a playout.

Unlike UCT, the NMCS does not explore the entire search space given enough time, but gains in precision as it explores the tree and is less prone to be stuck in a local maximum. This property led to generally better results from NMCS over UCT and other algorithms on optimization problems.

Results

Experimental setup

The aim of this work was to (i) generate large sets of novel molecules that expand the chemical diversity of available compound libraries, (ii) generate small molecules with drug-like properties, (iii) which in future could be used for VS campaigns and could be easily grown into larger drugs tailored for specific targets. As such, DrugSynthMC generates, in the absence of any training, SMILES of drug-like molecules without prior targets in mind. Thus, the score function is meant to only maximize the validity and drug-likeness of the output molecules and is not goal-oriented (e.g., tailored to bind a specific biological target). The drug-likeness is obtained by maximizing the general chemical properties associated with drugs according to the "Rule of 5" [40,41]. [42] Indeed, easily calculated physicochemical descriptors, such as molar mass and number of hydrogen bond donors and acceptors, have been found to correlate with the success rate of clinical trials. [43]

The function of compliance (score function) was defined as:

$$\alpha_1 = -max(mass - 500, 0)/500 \tag{1}$$

$$\alpha_2 = -max(natoms - 70, 0)/70 \tag{2}$$

$$\alpha_3 = min(natoms - 20, 0)/20 \tag{3}$$

$$\alpha_4 = -max(nhbd - 5, 0)/5 \tag{4}$$

$$\alpha_5 = -max(nhba - 10, 0)/10 \tag{5}$$

$$score = \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5 \tag{6}$$

With *mass* the molecular weight of the molecule in daltons, *natoms* the number of atoms (including hydrogens), *nhbd* the number of hydrogen bond donors, and *nhba* the number of hydrogen bond acceptors. This formula has the advantage of being computationally cheap and requiring only a pass through the SMILES string.

When generating molecules with a particular target in mind, this function can be used to produce drug-like molecules in tandem with a specific goal function.

To identify the most efficient method, we compared the UCT and NMCS MCS algorithms combined with different types of playouts:

1. random: the next character is selected uniformly randomly among the ones proposed by the search model.

2. enforced: the next character is selected uniformly randomly among the ones proposed by the search model. In order to generate compounds that are structurally valid and synthetically accessible, the score function aims to generate molecules containing the same ratios of heavy atoms (Nitrogen, Oxygen, Fluorine, Sulphur, Chlorine) as in FDA drugs (retrieved from the FDA subset of the ZINC20 database).[44]

3. ngram: the next character is selected randomly among the ones proposed by the search model, according to the conditional probabilities of the 3 characters ngram computed on the SMILES from the FDA database. To balance the sizes of the rings, cycles close with lengths 5, 6, and 7 with probabilities of 0.228, 0.751, and 0.020 respectively (computed from FDA data). No conditional probability was used to balance the type of ring (i.e., aromatic and aliphatic, homo- and hetero-cycles). Additionally, characters with a probability under 0.001 are pruned as they are judged too rare.

4. neural: the next character is selected randomly among the ones proposed by the search model, according to the neural network output weight given the incomplete current SMILES input. Additionally, characters with a probability under 0.001 are pruned as they are judged too rare.

For NMCS, we used a level of 3. For PUCT/UCT we used a constant of 1. We used PUCT instead of UCT when a prior was employed (ngrams and neural). PUCT and UCT used a timeout of 10 seconds, as PUCT is prone to locking itself in a local maximum even

with neural priors. All experiments were ran in Rust 1.59, on an Intel Core i7-11850H 2.50GHz using a single core.

Validation of the Generated Drug-like Molecules

In recent years, with the expansion of DL methods for drug design, several initiatives have been launched to assess generated compounds, which includes benchmarks such as Guacamol[45] and MOSES.[46] However, these benchmarks are not suitable for methods like DrugSynthMC that don't exclusively rely on training datasets. Instead, we evaluated similar metrics (validity, uniqueness, novelty, diversity, physicochemical properties, and synthesizability) and used comparable tools (RDkit, ZINC databases, AiZynthFinder) to validate our algorithm. To assess the reliability of the tool to generate valid and interpretable molecules, 10,000 generated SMILES were translated into structure representations using RDkit. In all cases, we see that RDkit is able to read and translate 100% of the inputted SMILES (Table 1), showing no syntax errors. This is significantly superior to other methods, which showed validity scores ranging from 85% for generative autoencoders to about 96% for RNN-based models[47,48].[49] The ability to generate novel compounds was determined by measuring the percentage of molecules in a library of 10,000 generated SMILES which was not present within ZINC-250K (containing nearly 250,000 molecules) (Table 2). In all cases, we see a high level of novelty. It is logical to assume that designing compounds based on general physicochemical properties of drugs instead of on a training set allows DL methods to explore a wider chemical space. Within each of the libraries generated, uniqueness was assessed by identifying the number of identical molecules (Table 3). This shows substantial differences among the different playouts used, with ngram and neural showing a higher number of replicated molecules within libraries. This is linked to the priors restricting the search space to what is more probable. Indeed, by determining the average edit distance as a measure of structural similarity of compounds (Table 4), we can see that both priors similarly restrict the explored chemical space (lower values indicating more similar structure, more

likely to have similar properties[50]). Conversely, by generating larger drugs with SMILES string containing 30 characters or more the uniqueness rises above 95% with all methods. While possible, it is more challenging to predict the synthesizability of larger drugs using retrosynthesis programs that search a synthetic route for a chosen molecule[51,52].

The first four rules of Lipinski are easy to compute. However, computing the logP of a molecule is not an easy task and is still an active research topic, with recent advancements using computationally expensive and state of the art neural approaches.[53] Faster approaches are available but they trade accuracy for speed. As such, to avoid making the evaluation function computationally expensive or inaccurate, the calculation of the logP was not included within the score function. It was instead subsequently computed on generated compounds using Rdkit which implements the atom-based Wildman-Crippen method.[54] In Table 5, we show this approach to be valid, as all molecules generated using the first 4 rules show logP values inferior to 5.

Using our model with the current parameters gives molecules that are completely drug-like in at least 99% of cases. Molecules with logP > 5 do not contain a cycle or subtrees (string-like) and are therefore easily identifiable. However as their logP is still very close to 5, they could nevertheless be considered drug-like.

In short, DrugSynthMC produces completely valid and novel drug-like molecules that show uniqueness in comparison to existing collections of drug-like molecules.

Table 1: Validity of 10 000 generated SMILES with different methods using RDkit sanitation

	random	enforced	ngram	neural
NMCS	100%	100%	100%	100%
PUCT	-	-	100%	100%
Sampling	100%	100%	100%	100%

Synthesizability of the Generated Drug-like Molecules

A recent comparison of tools used to predict the synthesizability of compounds carried out by Sanchez-Garcia et al.[55] showed that retrosynthesis programs tend to be more accurate

Table 2: Novelty of generated SMILES with different methods

	random	enforced	ngram	neural
NMCS	100%	99.99%	99.98%	99.96%
PUCT	-	-	99.99%	100%
Sampling	100%	100%	100%	99.99%

Table 3: Uniqueness of 10 000 generated SMILES with different methods

	random	enforced	ngram	neural
NMCS	99.94%	99.17%	81.78%	87.12%
PUCT	-	-	86%	85.32%
Sampling	99.83%	98%	82.58%	68.01%

Table 4: Average edit distance for 1 000 generated molecules compared 2 by 2 from each method

	random	enforced	ngram	neural
NMCS	17.930	14.266	12.954	13.988
UCT/PUCT	-	-	13.334	13.956
Sampling	17.801	13.940	13.100	14.051

Table 5: Proportion of logP < 5 of 10 000 generated SMILES with different methods

	random	enforced	ngram	neural
NMCS	100%	99.91%	100%	100%
PUCT	-	-	99.99%	100%
Sampling	99.98%	99.88%	99.98%	100%

than SA scores. Retrosynthesis programs use a combination of search algorithms and 1-step retrosynthesis deep learning to apply reactions to a molecule and divide it into reactants available on the market.

AiZynthFinder is an open-source full-fledged template-based retrosynthesis tool that uses MCTS, the same algorithm as UCT used here with slight differences. It is easily reproducible, and objective, but generates many false negatives as larger or too complex molecules can prove beyond the capability of the program to assess. According to the publication documenting the performance of AizynthFinder,[25] the overwhelming majority of successful paths to synthesis are found within 2 minutes of exploration. Indeed, this was consistent with our retrosynthesis analysis of drugs retrieved from the FDA subset, showing that AiZynthFinder finds routes for the majority of molecules within the first 2 min of search as seen in Figure 1. Hence, 2 min was chosen as maximum search time for the retrosynthesis assessment of the 1000 molecules generated.

There is a connection between mass of a molecule and success in identifying retrosynthesis synthetic routes, as seen in Figure **??**. We noticed that the larger the compounds generated the less likely AiZynthFinder successfully completes a search within 2 minutes or more. Indeed, AiZynthFinder only finds routes for about 60% of the 1615 FDA-approved within this time limit, despite 11.02% of the FDA molecules already being commercially available (Figure **??**). As this will not be the case for our molecules, their synthesizability rate is therefore expected to be under 50%. Therefore, we chose to generate lower molar mass drug-like compounds, at the expense of uniqueness, to show the ability of our approach to generate synthesizable compounds.

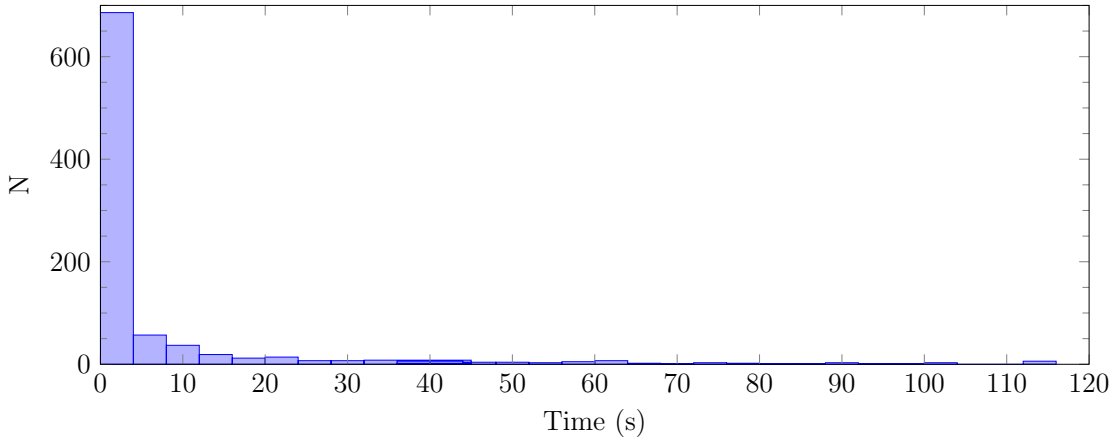


Figure 1: Search time in seconds for the 909 out of 1615 molecules from the FDA molecule database that AiZynthFinder found a route for

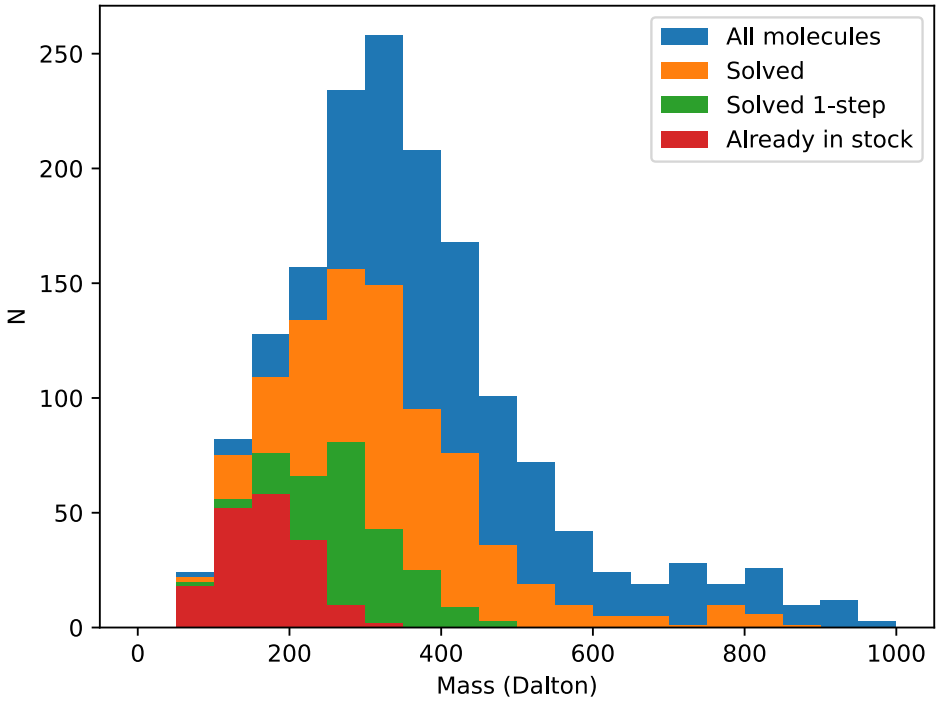


Figure 2: Amount of molecules AiZynthFinder found a route for in less than 2 minutes, separated in weight bins of 50 Daltons. "Solved 1-step" molecules are molecules whose synthesis route only has one step, they can be produced directly from commercially available compounds. "Already in stock" molecules are molecules from FDA that are immediately identified as commercially available by AiZynthFinder.

The relatively low success in finding retrosynthesis routes can be in part explained by the fact that AiZynthFinder uses the default 1-step retrosynthesis neural network trained on USPTO-50K incomplete data by AstraZeneca. However, as better performing 1-step retrosynthesis neural networks are not publicly available, results from AiZynthFinder are still useful to compare the predicted synthesizability of the compounds generated using the various methods used here.

In table 6 we show the percentage of synthesizability of the first 1000 molecules generated by each playout variant of the NMCS. Due to the similarities between the algorithms when using neural or ngram playouts, and, as shown above, the overall better performances of NMCS, we focused on the molecules generated by this latter method.

Table 6: Synthesizability of 1000 generated SMILES with NMCS with different playouts computed by AiZynthFinder in 2 minutes

	random	enforced	ngram	neural
NMCS	0.4%	0.9%	32.2%	18.1%

The ngram playout is the one that generates the largest number of synthesizable compounds. The 32.2% synthesizability rate is promising and in line with previously reported accuracy rates for different retrosynthesis programs.[56] The neural playouts, while still promising, provide a much lower rate of predicted synthesizability. This can be explained by the fact that the neural network does not return the exact conditional probability from the training set, and thus rare moves such as the shortcuts are over-represented in these generations. We adopted the neural network from Yang et al.,[29] with adaptations for our shortcuts and explicit bonds. While it showed worse outcomes with the implicit bonds and no shortcuts, it delivered similar results with the shortcuts. However, with further fine tuning, neural networks may have the potential to reach the same level of synthesizability achieved with the ngrams, although with the added disadvantage of having a slower execution (Table 7).

The random and enforced generations act as control experiments. As no policy governs the structure of the generation, nothing can direct the molecule generation toward a sen-

sible and synthesizable outcome. This results in a low rate of synthesizability. The 1-step retrosynthesis neural network used by AiZynthFinder is unable to propose reactions, thus ending the search long before the 2-minute time limit (generally in less than a second).

Physicochemical properties of the Generated Drug-like Molecules

To further validate drug-likeness of generated compounds, we compared the distributions of key physicochemical properties of molecules generated by NMCS with ngram playout (the method that generated the highest number of valid and synthesisable drugs-like compounds) with molecules generated with NMCS with random playout (used as control) and drugs retrieved from the FDA subset of ZINC-20,[44] abiding to the same general physicochemical properties we took into consideration in the design stage. It has been shown that to avoid reducing oral bioavailability, the number of hydrogen bond donors (HBD) should be lower than 6 and hydrogen bond acceptors (HBA) lower than 15,[57].[58] Despite the upper limitation of 10 and 5 for HBA and HBD, respectively, we found that compounds generated with ngram peak at 2 and 1 for HBA and HBD, respectively, with an overall lower number of HBA than random, and a higher ratio of compounds with fewer HBD than random and FDA, (Figures 3a 3b).

By design, ngram generates compounds with lower molar mass and total number of carbons (and consequently, lower number of heavy atoms) and hydrogens than drugs in FDA (Figures 3g 3d). This is only a side effect of stopping building the molecule once it respects all Lipinski rules. But it is also an indispensable requirement for (i) future optimization studies where compounds may need to be grown to adapt to pockets in targets and increase overall affinity, and (ii) synthesizability analysis with AiZynthFinder.

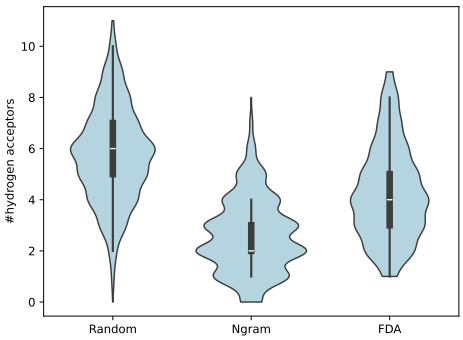
The distribution of heavy atoms is overall comparable in all plots, as the score function generates molecules containing the same heavy atoms (Nitrogen, Oxygen, Fluorine, Sulphur,

Chlorine) ratios as in FDA drugs (Figures 3n 3o 3l 3p 3h). DrugSynthMC score function balances the number and size of rings (Figures 3i 3j 3k), but not the type (aromatic and aliphatic, homo- and hetero-cycles), based on the probability of different rings occurrence calculated on FDA drugs. The formula returns drugs with a nearly equal distribution of compounds with zero or one aromatic cycle, which is lower than the aromatic cycle content in FDA drugs (Figure 3e). However, it has been shown that oral drugs with less than 3 aromatic rings have good compound developability,[59] suggesting that DrugSynthMC has the potential to generate compounds with a low risk of attrition in early-stage development. Generated compounds also have promising oral bioavailability parameters. Earlier work by Caminero Gomes Soares et al.[60] showed that FDA-approved drugs in the last 20 years have relatively stable numbers of rotatable bonds (mean 5, median 7.5) with about 89% drugs containing less than 10 rotatable bonds. Search with ngrams successfully generates compounds with less than 6 rotatable bonds (mean 3) and, in proportion, produced a higher number of molecules with fewer and higher rotatable bonds than the FDA drugs and Random playout, respectively, thus potentially identifying a balance between flexibility and diffusional cross-section (Fig. 2P).

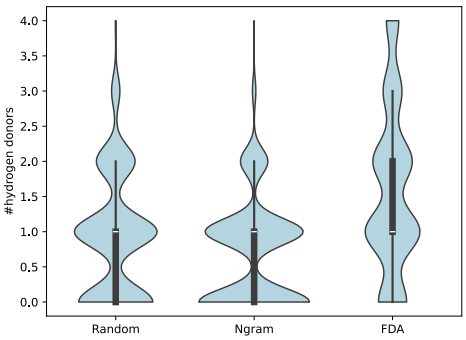
The results presented in Figure 3 should be analyzed while keeping in mind that the only goal of our approach is to generate novel, unique, synthesizable, drug-like molecules. It is not an attempt to create a new database with the exact same distribution of molecular descriptors as the FDA. It would however be possible to use our approach to do that by enforcing these molecular descriptors through the score function like it was done with enforced for the ratio of oxygen and nitrogen per carbon atoms.

Generation Times

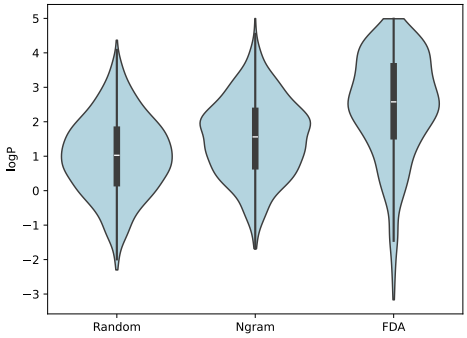
The capabilities of DrugSynthMC to reliably generate valid drug-like molecules are established in the previous sections. But generation time is an important metric to assess the usability of our approach, especially if it is to be used in tandem with target-based com-



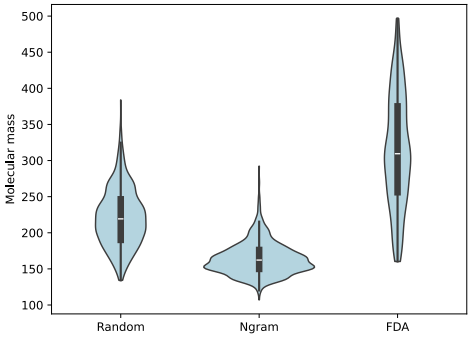
(a) #hydrogen acceptor



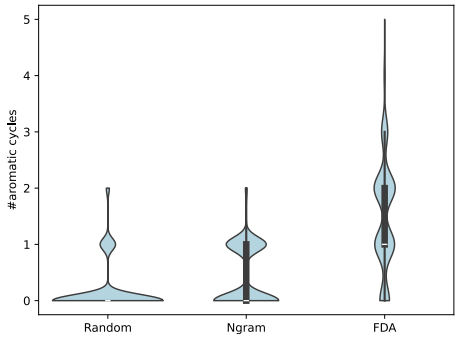
(b) #hydrogen donor



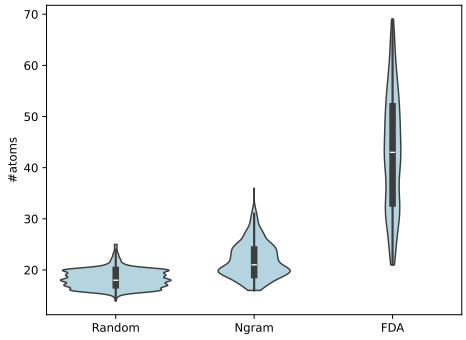
(c) logP



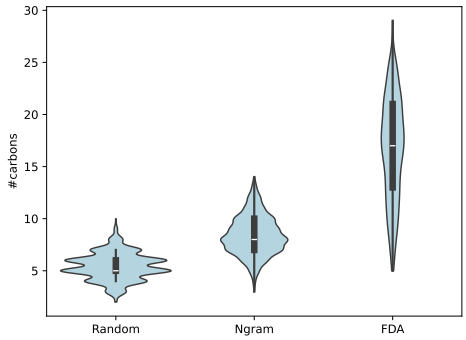
(d) g/mol



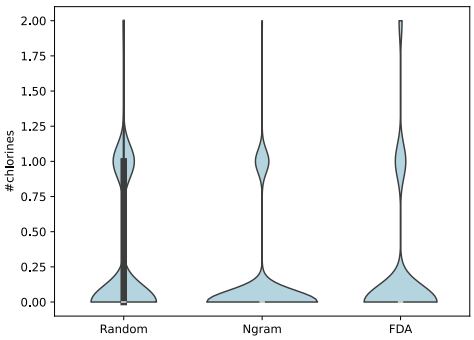
(e) #aromatic cycles



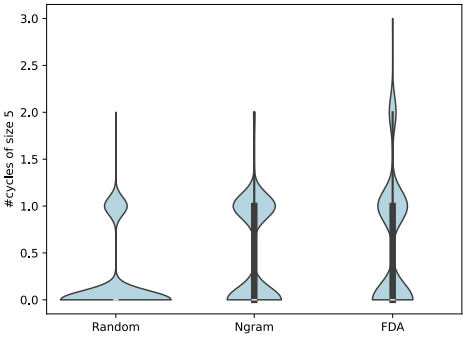
(f) #heavy atoms



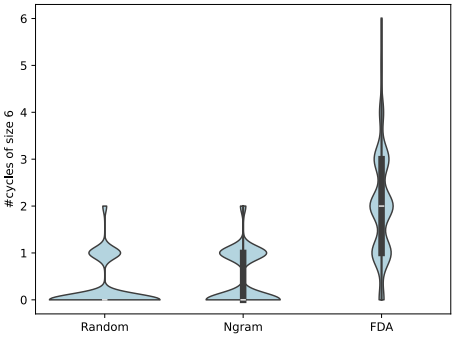
(g) #carbons



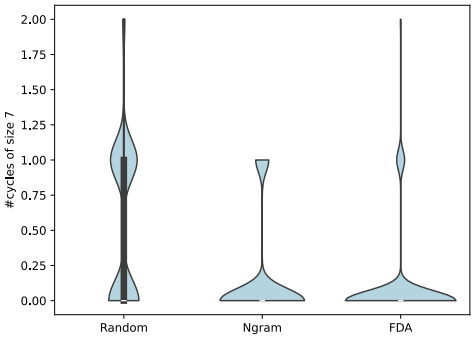
(h) #chlorines



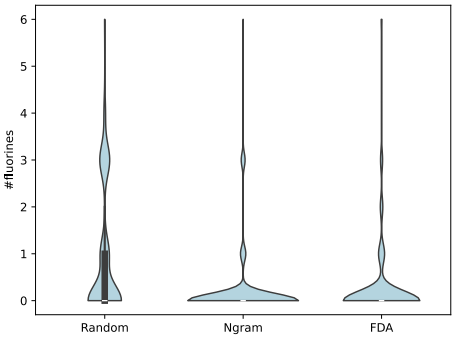
(i) #cycles of size 5



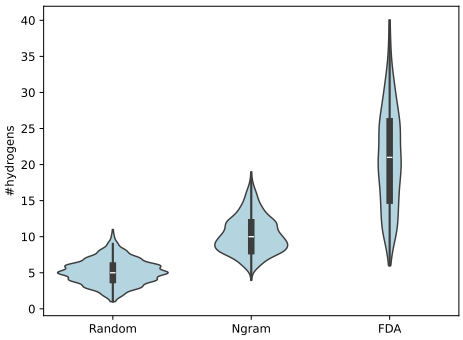
(j) #cycles of size 6



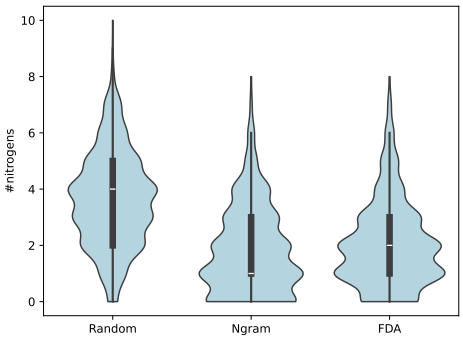
(k) #cycles of size 7



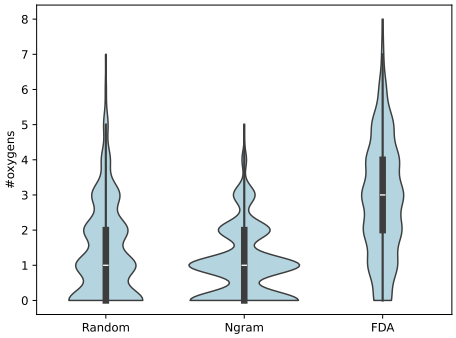
(l) #fluorines



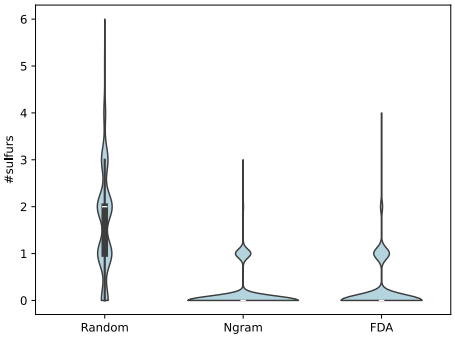
(m) #hydrogens



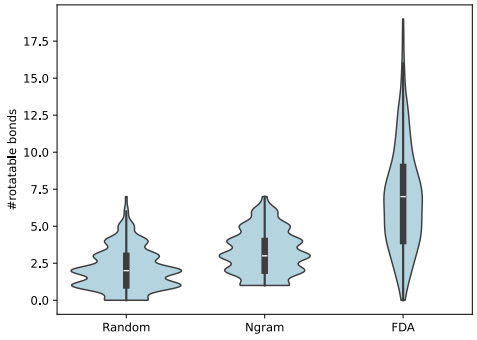
(n) #nitrogens



(o) #oxygens



(p) #sulfurs



(q) #rotatable bonds

Figure 3: Metrics of molecule generated using NMCS with ngram prior and random playouts compared to the FDA molecules

pounds optimization.

The time required to generate 1000 valid drug-like molecules in independent runs can be found in Table 7. This readout provides us with a few insights into our model, the search methods used, and the drug-likeness score function.

Table 7: Time to generate 1000 molecules with each method and algorithm in seconds

	random	enforced	ngram	neural
NMCS	1.366	9.395	0.753	3280.072
UCT/PUCT	-	-	0.507	2732.383
Sampling	9.154	675.544	0.482	2679.176

The random and enforced columns do not use a policy and show how the algorithm selection can affect the generation speed. Because it locks itself into local maxima induced by the shortcuts, UCT is unable to return Lipinski molecules (unless restarted immediately, turning it into Sampling). The shortcuts add a great amount of mass to the molecules and increase the score, but they are not the best overall choice to generate Lipinski molecules. Even with the priors, UCT locks itself into local maxima and requires restarts to generate Lipinski molecules. While we do not show these results in table 7, UCT without the shortcuts is able to return molecules for random and enforced generation. In this case, it is as fast as NMCS and Sampling over random generation and produces 1000 molecules in 0.60s. In contrast, it generates 1000 enforced molecules in 1440.00s seconds against 11.43s for NMCS and 212.77s for Sampling.

The NMCS shows a clear advantage over UCT and Sampling when the score function is hardened. It is a better optimization strategy than random playouts and does not lock itself in local maxima like UCT. The design of NMCS forces it to explore other subtrees of the search tree, thus preventing locking. This feature is also the reason why this approach is slower when using a prior in our experiments.

The prior columns for ngram and neural, give a different insight. The gap between the algorithms is closer because Sampling usually uses less than 20 playouts before finding a correct lipinski molecule. Both the ngram and neural priors are enough by themselves to

perform efficient playouts, returning Lipinski molecules. UCT and Sampling have similar outcomes because the small number of playouts undertaken do not let UCT reinforcement learning gather sufficient data before finding a suitable molecule. NMCS is however slightly slower due to the design that prevents it from locking itself in local maxima: because the algorithm explores other branches, possibly including ones not recommended by the prior, it can spend some time searching in less promising subtrees. This informs us that on problems that can be efficiently solved using playouts with a strong enough prior alone, NMCS can be outperformed by Sampling. However, we recommend using NMCS as it is usually better on harder problems, as shown by results shown in the columns for random and enforced playouts. We suspect that this state of play would extend to any specific goal-oriented generation, which is expected to be more complex than Lipinski molecule generation.

The column neural shows that the neural prior is much slower (approximately 5000 times) than the ngrams. This is the result of the difference in execution time between the neural network and the ngrams alone. With this instance of the neural network, this longer generation time is not rewarded with the generation of better molecules (as shown above).

Discussion

Our approach aims at generating character-by-character SMILES that can be defined as drug-like, through following the Lipinki rule of five. DrugSynthMC gives multiple insights on de novo drug-like molecule generation. First, it provides the unexpected result that ngrams can prove superior to the more advanced neural approaches. Second, it shows that usual MCTS algorithms such as UCT may be outperformed by other optimization algorithms. Finally, it delivers a significant improvement on the predicted synthesizability of the generated molecules as compared to previous published approaches.

AiZynthFinder was able to find retrosynthetic route for 32.2% of the molecules generated using ngrams in under 2 minutes. This result is promising, especially considering AiZyn-

thFinder found a route in under 2 minutes for only 909 out of the 1615 molecules of the FDA, despite using this dataset as part of its training.

It is possible that the ngrams were superior to the neural prior because they followed more closely the distribution. The superiority of NMCS over UCT in random and enforced generation was expected as NMCS is usually better than UCT on optimization problems.[39,61?] The easiness of generation when using priors both for UCT, NMCS, and Sampling was however unexpected, and caused the generation to show little differences between these algorithms. This means that both ngrams and neural network can efficiently learn molecule's structures to guide a search.

However, ngrams present limitations. They cannot learn nor use the context from the entire molecule. The search algorithm can make up for this under some circonstances but future advances in de novo generation might still require the use of neural networks. These could be used as prior to DrugSynthMC which we showed to be highly flexible and could easily be tuned to fit the need of future drug discovery projects. This would include the generation of customised drugs libraries tailored for specific binding pockets on bespoke targets.

Conclusion

While not focusing on set generation, DrugSynthMC is capable of generating sets that compete with state-of-the-art methods, with valid and drug-like molecules as outputs. In addition, a significant portion of these molecules are found to be synthesizable by retrosynthetic tools, while being completely novel.

Generating Lipinski molecules proved to be an easy task for Monte-Carlo algorithms, to the extent that using a prior does not differentiate between them. The next step of our research will focus on generating molecules with a harder goal in addition to the drug-likeness, such as the ability to bind chosen 3-dimensional pockets on selected biological targets. This

could be achieved by using a neural network trained on docking results as a score function to optimize molecules for specific binding sites.

Code and results are available at: https://github.com/RoucairolMilo/DrugSynthMC

Acknowledgement

References

(1) Zhang, H.; Chen, S. Cyclic peptide drugs approved in the last two decades (2001–2021). *RSC Chemical Biology* **2022**, *3*, 18–31.

(2) Zhavoronkov, A. Artificial intelligence for drug discovery, biomarker development, and generation of novel chemistry. 2018.

(3) Salmaso, V.; Moro, S. Bridging molecular docking to molecular dynamics in exploring ligand-protein recognition process: An overview. *Frontiers in pharmacology* **2018**, *9*, 923.

(4) Grinter, S. Z.; Zou, X. Challenges, applications, and recent advances of protein-ligand docking in structure-based drug design. *Molecules* **2014**, *19*, 10150–10176.

(5) Yuriev, E.; Holien, J.; Ramsland, P. A. Improvements, trends, and new ideas in molecular docking: 2012–2013 in review. *Journal of Molecular Recognition* **2015**, *28*, 581–604.

(6) Scior, T.; Bender, A.; Tresadern, G.; Medina-Franco, J. L.; Martínez-Mayorga, K.; Langer, T.; Cuanalo-Contreras, K.; Agrafiotis, D. K. Recognizing pitfalls in virtual screening: a critical review. *Journal of chemical information and modeling* **2012**, *52*, 867–881.

(7) Trezza, A.; Iovinelli, D.; Santucci, A.; Prischi, F.; Spiga, O. An integrated drug repurposing strategy for the rapid identification of potential SARS-CoV-2 viral inhibitors. *Scientific reports* **2020**, *10*, 13866.

(8) Bohacek, R. S.; McMartin, C.; Guida, W. C. The art and practice of structure-based drug design: a molecular modeling perspective. *Medicinal research reviews* **1996**, *16*, 3–50.

(9) Lyu, J.; Wang, S.; Balius, T. E.; Singh, I.; Levit, A.; Moroz, Y. S.; O'Meara, M. J.; Che, T.; Algaa, E.; Tolmachova, K.; others Ultra-large library docking for discovering new chemotypes. *Nature* **2019**, *566*, 224–229.

(10) Stein, R. M.; Kang, H. J.; McCorvy, J. D.; Glatfelter, G. C.; Jones, A. J.; Che, T.; Slocum, S.; Huang, X.-P.; Savych, O.; Moroz, Y. S.; others Virtual discovery of melatonin receptor ligands to modulate circadian rhythms. *Nature* **2020**, *579*, 609–614.

(11) Alon, A.; Lyu, J.; Braz, J. M.; Tummino, T. A.; Craik, V.; O'Meara, M. J.; Webb, C. M.; Radchenko, D. S.; Moroz, Y. S.; Huang, X.-P.; others Structures of the σ2 receptor enable docking for bioactive ligand discovery. *Nature* **2021**, *600*, 759–764.

(12) Sadybekov, A. A.; Sadybekov, A. V.; Liu, Y.; Iliopoulos-Tsoutsouvas, C.; Huang, X.-P.; Pickett, J.; Houser, B.; Patel, N.; Tran, N. K.; Tong, F.; others Synthon-based ligand discovery in virtual libraries of over 11 billion compounds. *Nature* **2022**, *601*, 452–459.

(13) Meyers, J.; Fabian, B.; Brown, N. De novo molecular design and generative models. *Drug Discovery Today* **2021**, *26*, 2707–2715.

(14) Bhisetti, G.; Fang, C. Artificial Intelligence–Enabled De Novo Design of Novel Compounds that Are Synthesizable. *Artificial Intelligence in Drug Design* **2022**, 409–419.

(15) Shiammala, P. N.; Duraimutharasan, N. K. B.; Vaseeharan, B.; Alothaim, A. S.; Al-Malki, E. S.; Snekaa, B.; Safi, S. Z.; Singh, S. K.; Velmurugan, D.; Selvaraj, C. Exploring the artificial intelligence and machine learning models in the context of drug design difficulties and future potential for the pharmaceutical sectors. *Methods* **2023**,

(16) Weininger, D. SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. *Journal of chemical information and computer sciences* **1988**, *28*, 31–36.

(17) Gómez-Bombarelli, R.; Wei, J. N.; Duvenaud, D.; Hernández-Lobato, J. M.; Sánchez-Lengeling, B.; Sheberla, D.; Aguilera-Iparraguirre, J.; Hirzel, T. D.; Adams, R. P.; Aspuru-Guzik, A. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science* **2018**, *4*, 268–276.

(18) Putin, E.; Asadulaev, A.; Ivanenkov, Y.; Aladinskiy, V.; Sanchez-Lengeling, B.; Aspuru-Guzik, A.; Zhavoronkov, A. Reinforced adversarial neural computer for de novo molecular design. *Journal of chemical information and modeling* **2018**, *58*, 1194–1204.

(19) Li, Y.; Zhang, L.; Liu, Z. Multi-objective de novo drug design with conditional graph generative model. *Journal of cheminformatics* **2018**, *10*, 1–24.

(20) Segler, M. H.; Kogej, T.; Tyrchan, C.; Waller, M. P. Generating focused molecule libraries for drug discovery with recurrent neural networks. *ACS central science* **2018**, *4*, 120–131.

(21) Coley, C. W.; Rogers, L.; Green, W. H.; Jensen, K. F. SCScore: synthetic complexity learned from a reaction corpus. *Journal of chemical information and modeling* **2018**, *58*, 252–261.

(22) Button, A.; Merk, D.; Hiss, J. A.; Schneider, G. Automated de novo molecular design by hybrid machine intelligence and rule-driven chemical synthesis. *Nature machine intelligence* **2019**, *1*, 307–315.

(23) Horwood, J.; Noutahi, E. Molecular design in synthetically accessible chemical space via deep reinforcement learning. *ACS omega* **2020**, *5*, 32984–32994.

(24) Bilodeau, C.; Jin, W.; Jaakkola, T.; Barzilay, R.; Jensen, K. F. Generative models for molecular discovery: Recent advances and challenges. *Wiley Interdisciplinary Reviews: Computational Molecular Science* **2022**, *12*, e1608.

(25) Genheden, S.; Thakkar, A.; Chadimová, V.; Reymond, J.-L.; Engkvist, O.; Bjerrum, E. AiZynthFinder: a fast, robust and flexible open-source software for retrosynthetic planning. *Journal of Cheminformatics* **2020**, *12*, 70.

(26) Rosin, C. D. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence* **2011**, *61*, 203–230.

(27) Silver, D. et al. Mastering the game of Go with deep neural networks and tree search. *Nature* **2016**, *529*, 484–489.

(28) Rosin, C. D. Nested rollout policy adaptation for monte carlo tree search. In IJCAI. 2011; pp 649–654.

(29) Yang, X.; Zhang, J.; Yoshizoe, K.; Terayama, K.; Tsuda, K. ChemTS: an efficient python library for de novo molecular generation. *Science and technology of advanced materials* **2017**, *18*, 972–976.

(30) Metropolis, N.; Ulam, S. The monte carlo method. *Journal of the American statistical association* **1949**, *44*, 335–341.

(31) Kocsis, L.; Szepesvári, C. Bandit Based Monte-Carlo Planning. Machine Learning: ECML 2006. Berlin, Heidelberg, 2006; pp 282–293.

(32) Deng, H.; Yuan, X.; Tian, Y.; Hu, J. Neural-augmented two-stage Monte Carlo tree search with over-sampling for protein folding in HP Model. *IEEJ Transactions on Electrical and Electronic Engineering* **2022**, *17*, 685–694.

(33) Cazenave, T.; Fournier, T. Monte Carlo Inverse Folding. Monte Carlo Search at IJCAI. 2020.

(34) Cazenave, T.; Touzani, H. Monte Carlo Inverse RNA Folding. RNA Design. Methods in Molecular Biology. Humana Press. 2024.

(35) Cazenave, T. Learning a Prior for Monte Carlo Search by Replaying Solutions to Combinatorial Problems. arXiv:2401.10431. 2024.

(36) Präntare, F.; Appelgren, H.; Heintz, F. Anytime heuristic and monte carlo methods for large-scale simultaneous coalition structure generation and assignment. Proceedings of the AAAI Conference on Artificial Intelligence. 2021; pp 11317–11324.

(37) Saffidine, A.; Cazenave, T.; Méhat, J. UCD: Upper Confidence Bound for Rooted Directed Acyclic Graphs. *Knowledge-Based Systems* **2011**, *34*, 26–33.

(38) Dang, C.; Bazgan, C.; Cazenave, T.; Chopin, M.; Wuillemin, P.-H. Monte carlo search algorithms for network traffic engineering. Joint European Conference on Machine Learning and Knowledge Discovery in Databases. 2021; pp 486–501.

(39) Roucairol, M.; Cazenave, T. Refutation of Spectral Graph Theory Conjectures with Monte Carlo Search. COCOON 2022. 2022.

(40) Lipinski, C. A. Drug-like properties and the causes of poor solubility and poor permeability. *Journal of pharmacological and toxicological methods* **2000**, *44*, 235–249.

(41) Lipinski, C. A.; Lombardo, F.; Dominy, B. W.; Feeney, P. J. Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings. *Advanced drug delivery reviews* **2012**, *64*, 4–17.

(42) Doak, B. C.; Kihlberg, J. Drug discovery beyond the rule of 5-Opportunities and challenges. *Expert Opinion on Drug Discovery* **2017**, *12*, 115–119.

(43) Takács, G.; Sándor, M.; Szalai, Z.; Kiss, R.; Balogh, G. T. Analysis of the uncharted, druglike property space by self-organizing maps. *Molecular Diversity* **2021**, 1–15.

(44) Irwin, J. J.; Tang, K. G.; Young, J.; Dandarchuluun, C.; Wong, B. R.; Khurelbaatar, M.; Moroz, Y. S.; Mayfield, J.; Sayle, R. A. ZINC20—a free ultralarge-scale chemical

database for ligand discovery. *Journal of chemical information and modeling* **2020**, *60*, 6065–6073.

(45) Brown, N.; Fiscato, M.; Segler, M. H.; Vaucher, A. C. GuacaMol: benchmarking models for de novo molecular design. *Journal of chemical information and modeling* **2019**, *59*, 1096–1108.

(46) Polykovskiy, D.; Zhebrak, A.; Sanchez-Lengeling, B.; Golovanov, S.; Tatanov, O.; Belyaev, S.; Kurbanov, R.; Artamonov, A.; Aladinskiy, V.; Veselov, M.; others Molecular sets (MOSES): a benchmarking platform for molecular generation models. *Frontiers in pharmacology* **2020**, *11*, 565644.

(47) Krishnan, S. R.; Bung, N.; Vangala, S. R.; Srinivasan, R.; Bulusu, G.; Roy, A. De novo structure-based drug design using deep learning. *Journal of Chemical Information and Modeling* **2021**, *62*, 5100–5109.

(48) Handa, K.; Thomas, M. C.; Kageyama, M.; Iijima, T.; Bender, A. On the difficulty of validating molecular generative models realistically: a case study on public and proprietary data. *Journal of Cheminformatics* **2023**, *15*, 112.

(49) Schoenmaker, L.; Béquignon, O. J.; Jespers, W.; van Westen, G. J. UnCorrupt SMILES: a novel approach to de novo design. *Journal of Cheminformatics* **2023**, *15*, 22.

(50) Garcia-Hernandez, C.; Fernandez, A.; Serratosa, F. Ligand-based virtual screening using graph edit distance as molecular similarity measure. *Journal of chemical information and modeling* **2019**, *59*, 1410–1421.

(51) Coley, C. W.; Green, W. H.; Jensen, K. F. Machine learning in computer-aided synthesis planning. *Accounts of chemical research* **2018**, *51*, 1281–1289.

(52) Engkvist, O.; Norrby, P.-O.; Selmi, N.; Lam, Y.-h.; Peng, Z.; Sherer, E. C.; Amberg, W.;

Erhard, T.; Smyth, L. A. Computational prediction of chemical reactions: current status and outlook. *Drug discovery today* **2018**, *23*, 1203–1218.

(53) Tang, B.; Kramer, S. T.; Fang, M.; Qiu, Y.; Wu, Z.; Xu, D. A self-attention based message passing neural network for predicting molecular lipophilicity and aqueous solubility. *Journal of cheminformatics* **2020**, *12*, 1–9.

(54) Wildman, S. A.; Crippen, G. M. Prediction of physicochemical parameters by atomic contributions. *Journal of chemical information and computer sciences* **1999**, *39*, 868–873.

(55) Sanchez-Garcia, R.; Havasi, D.; Takács, G.; Robinson, M. C.; von Delft, F.; Deane, C. M.; others CoPriNet: graph neural networks provide accurate and rapid compound price prediction for molecule prioritisation. *Digital Discovery* **2023**, *2*, 103–111.

(56) Jiang, Y.; Yu, Y.; Kong, M.; Mei, Y.; Yuan, L.; Huang, Z.; Kuang, K.; Wang, Z.; Yao, H.; Zou, J.; Coley, C. W.; Wei, Y. Artificial Intelligence for Retrosynthesis Prediction. *Engineering* **2023**, *25*, 32–50.

(57) Doak, B. C.; Over, B.; Giordanetto, F.; Kihlberg, J. Oral druggable space beyond the rule of 5: insights from drugs and clinical candidates. *Chemistry & biology* **2014**, *21*, 1115–1142.

(58) Doak, B. C.; Zheng, J.; Dobritzsch, D.; Kihlberg, J. How beyond rule of 5 drugs and clinical candidates bind to their targets. *Journal of medicinal chemistry* **2016**, *59*, 2312–2327.

(59) Ritchie, T. J.; Macdonald, S. J. The impact of aromatic ring count on compound developability–are too many aromatic rings a liability in drug design? *Drug discovery today* **2009**, *14*, 1011–1020.

(60) Caminero Gomes Soares, A.; Marques Sousa, G. H.; Calil, R. L.; Goulart Trossini, G. H. Absorption matters: A closer look at popular oral bioavailability rules for drug approvals. *Molecular informatics* **2023**, *42*, e202300115.

(61) Roucairol, M.; Cazenave, T. Solving the Hydrophobic-Polar Model with Nested Monte Carlo Search. International Conference on Computational Collective Intelligence. 2023; pp 619–631.

Chapter 4

Puzzle design

4.1 Nonograms

I am an avid player of nonograms. I cannot get enough of them. Unfortunately, each nonogram game only comes with one or two hundred puzzles. And to the best of my knowledge, no game proposes endless procedurally generated nongrams or alternated game modes (like versus, time attack, or survival).

This project aimed at producing a new way to evaluate nonograms using a solver, and generate nonograms according to their evaluations. Evaluating the fun proved tricky, it ended up only helping to avoid generating unfun nonograms. However, the difficulty was more accurately evaluated. The software was able to produce nonograms exceeding human capacities.

This problem was not useful as a benchmark for comparing MCTS algorithms, they performed similarly.

Generating Difficult and Fun Nonograms

Milo Roucairol and Tristan Cazenave

LAMSADE, Université Paris Dauphine - PSL
Place du Maréchal de Lattre de Tassigny, 75016 Paris

Abstract. Nonograms are Japanese logic puzzles where the player must find a 2D black-and-white image using information on the columns and rows. Here we present a new method to generate nonograms of varying difficulties and enjoyability using a human-like solver to estimate the difficulty of a puzzle, and Monte Carlo Tree Search algorithms to optimize the estimation of the difficulty.

Keywords: Nonogram · Monte Carlo · puzzle.

1 Introduction

Logic puzzles are a staple of the gaming landscape. Widespread as early as 1913 thanks to newspaper, or even in 1783 with Euler's Latin square. Every living person has encountered one (be it crossword or sudoku), and likely solved one in their life. They are especially popular among retired people and computer scientists, as they are very similar to many computer science logic problems. Solving or programming a solver for these problems is common in CS studies and research.

Sudoku and Sokoban are two of the most notable logic puzzles of the 20th century. Another notable Japanese logic puzzle from the 80's was the Nonogram. It was quickly taken over by Nintendo with the Picross series for the Game Boy and the SNES with Mario's Picross in 1995 and entries using the sprites from Nintendo's most popular franchises. Spinning off multiple variants, like color nonograms, mosaic nonogram, nonogram with unknowns, mega-picross etc. We are interested in the original version of the nonograms.

A nonogram is a grid-based logic puzzle. The player has to fill a grid with either black or white squares until there are no more unknown cells and the picture is fully formed. The player is given instructions on the rows and columns in the form of numbers on top and left of the grid. Each number n indicates that there are exactly n adjacent black squares. The order in which the numbers appear is important too, the groups of adjacent black squares mus appear in the same order as on the indication.

Here we set out to generate Nonogram puzzles of varying difficulties, this has been the subject of some research. First in 2009, K. Joost Batenburg et al. [1] introduced a method to evaluate the difficulty and generate simple nonograms. Followed in 2012 by another assessment of the difficulty of nonograms [2]. However both these research do not focus on a player related difficulty but only to a

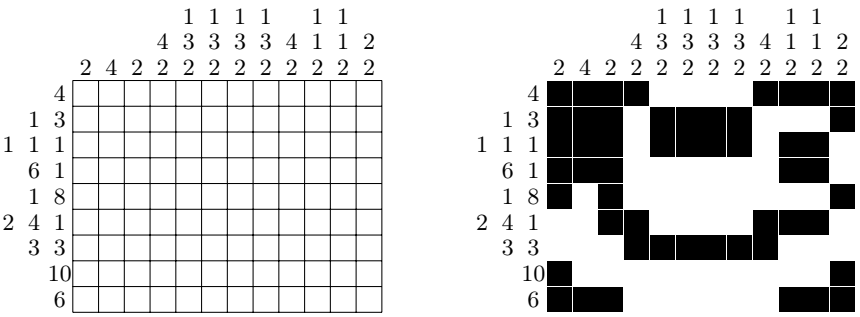


Fig. 1: Example of a medium sized nonogram, it depicts a cup of coffee on a saucer

solver/mathematical related difficulty, here we intend to explore the nonogram design with human players in mind. Solvers for nonograms are also another point of interest, with genetic approaches like in Wiggers' work [12].

In this paper we will use a human-like deterministic solver to evaluate the difficulty and enjoyability (fun) of the generated nonograms, and state of the art Monte Carlo Search algorithms for the generation process. Our goal is to make a fast nonogram generator with targetable difficulty to allow for new ways of playing nonograms: time attack, survival, or endless modees, with a set or increasing difficulties (Tetris-like gameplay).

2 "Human" Solver

An optimization process requires a way to evaluate the quality of a state. A state of our search tree is small a black-and-white image. We imagine two ways to evaluate the difficulty and enjoyability of that image:

1. A neural network trained on a set of images and their evaluation by humans
2. A solver that emulates human logic and stores data about the resolution

We decided to use a solver because designing a human-like solver is an interesting and intuitive task, and the neural network training datasets are not publicly available yet.

One technique that is commonly used when solving nonograms is to check the possible extreme positions of a group of adjacent black tiles and see if they overlap. If they do, the part where they overlap is necessarily black. This is the main technique used by humans, as such it is also the main one used by the solver. Conversely, spaces where no possible conformation had a black square on add a white square

In Figure 2, first the 8 white cells are deduced given all positions of the 10 consecutive white cells overlap on these 8 cells. Then, all possible positions of the 2 in the third column exclude the 6 first cells from top to bottom (denoted

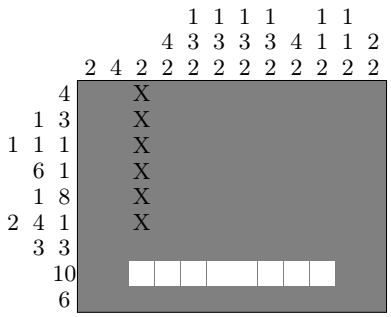


Fig. 2: Example of column and row swipes

with black "X"). These 6 cells must be black. This in turn helps us fill the 1st, 4th, 5th and 6th lines from the top.

Our method takes a line or columns, it's indications, and the already-filled cells, and compute all possible fillings that meet the requirements of the indications. Then if for a cell in this line, all possible filling leave it black or white, the cell is filled in the grid. This method is called over the rows and columns according to the last cells modified, this mimics human solving as the attention and memory of a human player are focused on the last cells modified too.

This simple method is enough to solve most nonograms, but the hardest ones may require a more advanced techniques, like the one called "edge solving". Edge solving is a type of deterministic guessing, it is done by guessing a white square, usually on the edges of the grid, and unrolling the usual resolution until a contradiction is found. Once a contradiction is found, the cell that was guessed as white is colored black. That technique is unknown by most players, and should be avoided unless the difficulty target is high and it only happens 1 to 3 times in the puzzle.

The algorithm has a stack of rows and lines to check (starting with all rows and columns), it checks them all until if finds a line/row move, it then applies that move to the row or columns and add the rows and columns of all modified cells to the stack. If the algorithm does not find a line/row move, it tries to find a deterministic guessing move that gives a contradiction. If no move is found it returns the grid in its state, completed or not. We decided to not allow the solver to do another deterministic guess while one is active, thus used as a score function it does not produce nonograms unsuitable for humans. This is not a problem since deterministic guessing is almost never used in most nonogram games, and never in a nested fashion to the best of our knowledge.

The solving time of the algorithm is a metric sufficient to design hard nonograms by maximizing it. But for more precise difficulty targets, the solver returns other metrics such as the number of time it used edge solving, the number of step, the length of each backtracking, and others. These metrics are used to estimate the fun and the difficulty of a puzzle.

3 Generation

3.1 Search Space

The generation process takes the form of a search tree search as we use Monte-Carlo Tree Search and other deterministic search algorithms. All the algorithms share the same search tree (or search space). The root of the search tree is the initial nonogram that is submitted to the algorithm: an empty grid with black cells only. From that state, the available moves are swapping the value (white or black) of any rectangle inside the grid.

This simple single move allows the search algorithms to reach diverse grids in fewer moves than setting the value of each cell. It also allows for drastic changes in difficulty and grid layout in very few moves, while also setting the value of each cell individually if needed.

It is possible to start from an already existing nonogram, or from an image, and input a list of cells that cannot have their value swapped to preserve the depiction: nonograms usually depict objects.

3.2 Algorithms

Monte Carlo search algorithms are a family of search algorithms that use sampling (playouts) to learn about a search space to guide the search. They are a key element of recent breakthrough in game playing such as Deepmind's AlphaGo [11], but are also widely used in other machine learning applications like multi-step retrosynthesis [5] or graph theory [9]. We selected the following algorithms.

1. UCT: Upper Confidence bound applied to Trees, the most commonly used MCTS algorithm. [6]
2. RAVE: Rapid Action Value Estimation, a variant of UCT using All Moves As First (AMAF), a machine learning technique appropriate to our search model because the order of the moves does not matter. [4]
3. NMCS: Nested Monte Carlo Search, a method that recursively calls lower level version of themselves on child states to find the route leading to the best score. A level 0 NMCS is a playout. [3]
4. LNMCS; Lazy Nested Monte Carlo Search, a variant of NMCS that introduces the exploitation/exploration dilemma and prunes lower level LNMCS using playout based evaluation. [10]
5. NRPA: Nested Rollout Policy adaptation, an algorithm inspired by NMCS that learns online a policy to guide the playouts. [8]

For combinatorial optimization, two families of MCS exist. The main and most commonly used one is called MCTS and uses an iterative approach, it includes UCT and RAVE. The other family is recursive, it includes NMCS, LNMCS, and NRPA.

Deterministic search algorithms are simpler search algorithms that do not involve sampling or machine learning. Intuitively we could expect them to provide

inferior results compared to methods using machine learning. Recent approaches show that this is not always the case, they can outperform even the most complex state-of-the-art machine learning on certain problems, as shown in [7,9]. We selected two widely known deterministic search algorithms:

1. BFS: Best First Search uses a list to keep track of the best candidates, it opens the best one, evaluates its child, inserts them in the list according to their evaluation, and repeats. This algorithm is complete, it will explore the entire search space given enough time.
2. BEAM: Beam Search keeps width w states open at all depth. From a depth n it opens all the children of the w states and keeps the w best ones for depth $n + 1$.

4 Results

4.1 Experimental Setup

The experiments were made using a 2.60 GHz i5-13600K Intel single core.

We define two goals of optimization:

1. Difficulty for the solver program: time spent solving it by the solver.
2. Difficulty for players: estimation by the solver.
3. Fun: estimation by the solver.

Nonograms also come in many different sizes, we decided to compare our program on sizes 5x10, 10x10, and 15x15 as these are the most common nonogram sizes used in games. Larger nonograms exist too, but the focus is usually on the picture depicted and not on the difficulty or the puzzling aspect.

Monte Carlo Search experiments require multiple runs. We observed an important standard deviation among preliminary runs, so we decided to run each algorithm 10 times over each combination of size and goal.

Finally, this method is intended to be used in real-time by games for player versus player, or survival (like Tetris for example) game modes for example. In these game modes small new nonograms must be generated as fast as the player solves them so we decided to set the optimization time to 60 seconds on an Intel Core i5-13600K using a single core. This method is also fit for generating regular low-quality nonogram games, or daily challenges.

For Hyperparameters, BEAM uses a width of 10, UCT a learning ratio of 1, RAVE a treshold of 5, NMCS a level of 2, NRPA a level of 2, and LNMCS a level of 3, pruning ratio of 0.8 and 3 playouts per estimation.

4.2 Optimizing the Solver Difficulty

The solver was made to behave like a human, thus the time taken to solve a nonogram is indicative of its difficulty, but it is also an interesting task by itself as a benchmark for estimating MCS algorithms performances. The solving times of graphs optimized to be more complex to the solver by all algorithms are presented in Table 1.

	BFS	BEAM	UCT	RAVE	NMCS	LNMCS	NRPA
5x5	0.000417	0.000181	0.00150	0.00121	0.00179	0.00180	0.000131
5x10	0.0200	0.00219	0.226	0.145	0.103	0.0963	0.00272
10x10	0.0221	0.00305	0.591	0.568	0.308	0.563	0.0286
15x15	0.00995	0.00254	9.578	10.466	3.066	13.07	1.741

Table 1: Mean solving times of nonograms generated in 60s in seconds, greater values indicate success from the Monte Carlo optimization algorithm

4.3 Optimizing the Estimated Player Difficulty

To estimate the difficulty for a human player, we focus on three metrics:

1. The number of steps required to solve the puzzle
2. The number of times edge guessing is required
3. The number of times backtracking is required (the next available move is not in vicinity)

Only focusing on the number of steps like in [1] would not result in an actually difficult puzzle, but in tedious ones (like the one shown in Figure 4 of their paper). While the difficulty and tediousness often overlap in nonograms, and a minimum number of steps is required to make a hard enough puzzle, the number of steps is not sufficient to build difficult nonograms, and is here the least important part of the evaluation. The other two members of the difficulty estimation are the number of times the player must backtrack to find a new available move (the move is N cells away from any modified cells last), this is the most important part of the equation and where we think resides the true difficulty. The last part of the equation is the number of advanced deterministic guessing techniques the player must use (edge guessing), puzzles must refrain from using too many of these in order to remain enjoyable, as such only the reward follows a square root function.

$$difficulty(N) = n(N) + \sqrt{g(N)} * 50 + d(N) * 10$$

Where N is the solution to the nonogram, $n(N)$ the number of steps in that solution, $g(N)$ the number of deterministic guessing, and $d(N)$ the number of times no move is found in the 7 lines and rows checked after making a move. The factors (50 and 10) were set to these values because a long backtracking is approximately 10 times more complex and time consuming for a human than executing moves found directly. A deterministic guessing can be simple or very hard, for the sake of the method's simplicity it is set 50, but could be computed more accurately depending on what happens during the guessing, or set to higher values like 100 by default.

The estimated difficulty scores of graph optimized by all algorithms are presented in Table 2

	BFS	BEAM	UCT	RAVE	NMCS	LNMCS	NRPA
5x5	103.710	103.710	158.670	148.930	168.956	166.241	96.536
5x10	112	112	197.533	184.609	211.582	212.475	116.202
10x10	181.602	217	277.778	280.949	271.944	272.314	198.370
15x15	299	401	308.975	313.443	320.596	296.749	282.483

Table 2: Mean estimated difficulty scores of nonograms generated in 60s

4.4 Optimizing the Fun

Estimating the human player's fun is a harder task as it is even more subjective. According to players, the fun is "like dominos or finishing a jigsaw puzzle, seeing something meticulously set up and solved line by line finally stepping back to look at the result, it's satisfying because of completion itself", or "solving any nonogram gives me a dopamine hit, but the bigger ones definitely hit harder.". Other players find satisfaction in hard puzzles (but we are not going to use this definition of fun in this section), otherwise the puzzles are generally fun by default, unless they are unfun.

An unfun puzzle can be defined as unnecessarily tedious and frustrating. As such, the number of total steps, edge guessing, and backtracking must be limited with some tolerance to avoid making the puzzles so simple they become tedious again.

To avoid puzzles so easy they become unfun, one of the terms in the parenthesis favors balance between the number of black and white cells. The other is the kurtosis value of the lengths of the moves of the solution. Both are multiplied by the number of unique lengths of the moves of the solution because players enjoy the most variety.

$$fun(N) = (\frac{4 * b(N)w(N)}{s(N)^2} - k(N)) * u(N) + 5 - max(d(N), 5) - g(N)^2$$

Where N is the solution to the nonogram, $s(N)$ the size of the grid (number of cells), $k(N)$ the kurtosis of the lengths of the moves used, $u(N)$ the number of moves of different lengths used, $b(N)$ the number of black cells, $w(n)$ the number of white cells, $n(N)$ the number of steps in that solution, $g(N)$ the number of deterministic guessing, and $d(N)$ the number of times no move is found in the 7 lines and rows checked after making a move. Like with the difficulty, this function is an attempt to model the fun. It is to be improved, ideally supported by puzzles rated by players, which we do not have.

The estimated fun scores of graph optimized by all algorithms are presented in Table 3

4.5 Overview of the Solver's Estimations

To assess our estimation function and solver, we ran it on 150 nonograms, in figure 3 we show 15 of them and their fun scores, difficulty scores, and solving times.

	BFS	BEAM	UCT	RAVE	NMCS	LNMCS	NRPA
5x5	12.765	12.765	12.765	12.696	12.765	12.765	11.099
5x10	14.008	0	16.523	15.642	16.959	17.072	14.326
10x10	18.596	0	19.807	18.540	20.213	19.888	16.417
15x15	18.976	0	13.577	12.113	11.663	13.235	16.225

Table 3: Mean estimated fun scores of nonograms generated in 60s

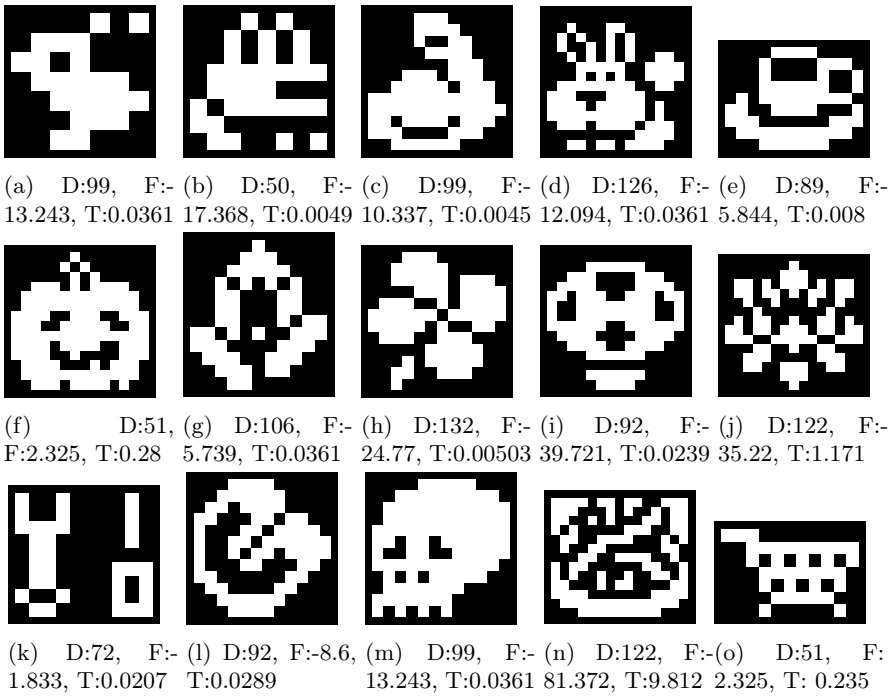
5 Discussion

With the optimization of the solver's time on Table 1: finding nonograms complex enough to maximize the time spent by the solver solving it, the size 15x15 is too large for most algorithms and does not allow to collect reliable data as the solver is rapidly countered by UCT, RAVE and LNMCS, but not NMCS. The standard deviation is high enough that LNMCS performance can be attributed to luck (UCT 5.019, RAVE 3.550, LNMCS 8.323, NMCS 2.544, NRPA 0.869). But this does not indicate that the solver is underperforming as seen in Figure 3 where it solves most nonograms rapidly. On lower grid sizes, LNMCS, UCT and RAVE are similar, with standard deviations of approximately 0.3 on 10x10. NMCS and NRPA are outperformed by the other MCS algorithms. BFS and BEAM results are even lower, this may be due to the use of playouts wich produce complex puzzles without the need to evaluate at each step (MCS algorithms only evaluate at the end), leading the deterministic algorithms to be left behind with larger grids. However, deterministic algorithms seem to perform better on small grids.

The optimization of the estimation of difficulty with a handmade function on Table 2 shows most MCS algorithms providing similar scores. The standard deviation is approximately 30 with all the MCS algorithms for size 15x15. Differences between UCT, RAVE, NMCS, and LNMCS are minimal and can be attributed to their random nature. However, NRPA, the only algorithm learning a policy, underperforms, while the BEAM search outperforms all MCS algorithms. The proximity of the 10x10 and 15x15 difficulty scores for the MCS algorithms seem to indicate that the solving times become too important and block the algorithms at the beginning of the search trees. BEAM evaluates the children of 10 states and goes down the search tree, thus allowing it to reach a harder puzzle.

The optimization of the estimation of fun with a handmade function on Table 3 show that BEAM locks itself in a local maximum. Again the MCS algorithms share similar results, with a slight advantage for LNMCS, they all have a standard deviation of approximately 1 for sizes 10x10 and 15x15 and less than 0.5 for sizes 5x5 and 10x5. The 15x15 size is where our optimization process starts to struggle to optimize the fun further, it seems adequate since the tediousness of a puzzle increases faster than its fun with the size in our opinion.

The application of the evaluations to select 15 nonograms in Figure 3 shows that the difficulty score, while far from being perfect, is quite correlated to the difficulty of the puzzles and is sometimes better at evaluating the difficulty than the solver time. However, our attempt at a fun function is able to produce fun



(a) D:99, F:-13.243, T:0.0361 (b) D:50, F:-17.368, T:0.0049 (c) D:99, F:-10.337, T:0.0045 (d) D:126, F:-12.094, T:0.0361 (e) D:89, F:-5.844, T:0.008

(f) D:51, F:2.325, T:0.28 (g) D:106, F:-5.739, T:0.0361 (h) D:132, F:-24.77, T:0.00503 (i) D:92, F:-39.721, T:0.0239 (j) D:122, F:-35.22, T:1.171

(k) D:72, F:-1.833, T:0.0207 (l) D:92, F:-8.6, T:0.0289 (m) D:99, F:-13.243, T:0.0361 (n) D:122, F:-81.372, T:9.812 (o) D:51, F: 2.325, T: 0.235

Fig. 3: Solver and its estimations applied to 15 nonograms, D: difficulty score, F: fun score, T: solving time by the solver in seconds

puzzles when used as a goal function for optimization but fails to recognize the fun in these nonograms. This may be explained by the penalization of deterministic guessing, and the kurtosis, for example Figure 3n has a high kurtosis as it has a mean of move length of 2, same for the kurtosis of Figure 3j. The imbalance between black and white, which does not necessarily have to be respected in handcrafted puzzles, can skew the estimation of fun too. The fun evaluation function aims to avoid tedious puzzles, even if it may produce false negatives.

6 Conclusion

In this paper we provided two new ways of evaluating the difficulty of Nonogram puzzles, improving on the previous work. These two new ways being using the solving time of a solver deigned to tackle the puzzle in approximately the same way as a human, and a new difficulty function taking backtracking and deterministic guessing into account. We also experimented with the definition of fun, with a mixed success, fun is highly subjective and our method only avoids unfun puzzles, but can flag fun puzzles as unfun, partly due to assumption on the fun definition which is not the same for all players.

We then used these functions to design new nonogram puzzles using search algorithms including State of The Art Monte Carlo Search algorithms. While MCS algorithms offered similar results, we noticed that NRPA was either able to learn a policy and outperform the others, or be outperformed.

Our method is apt for this task and can generate many nonograms of target difficulty rapidly. You can find the code here:

https://github.com/RoucairolMilo/nonogram

References

1. Batenburg, K.J., Henstra, S., Kosters, W.A., Palenstijn, W.J.: Constructing simple nonograms of varying difficulty. Pure Mathematics and Applications (Pu. MA) **20**, 1–15 (2009)
2. Batenburg, K.J., Kosters, W.A.: On the difficulty of nonograms. ICGA journal **35**(4), 195–205 (2012)
3. Cazenave, T.: Nested Monte-Carlo Search. In: Boutilier, C. (ed.) IJCAI. pp. 456–461 (2009)
4. Gelly, S., Silver, D.: Monte-carlo tree search and rapid action value estimation in computer go. Artif. Intell. **175**(11), 1856–1875 (2011). https://doi.org/10.1016/j.artint.2011.03.007, https://doi.org/10.1016/j.artint.2011.03.007
5. Genheden, S., Thakkar, A., Chadimová, V., Reymond, J.L., Engkvist, O., Bjerrum, E.: AiZynthFinder: a fast, robust and flexible open-source software for retrosynthetic planning. Journal of Cheminformatics **12**(1), 70 (Dec 2020). https://doi.org/10.1186/s13321-020-00472-1, https://jcheminf.biomedcentral.com/articles/10.1186/s13321-020-00472-1
6. Kocsis, L., Szepesvári, C.: Bandit based Monte-Carlo planning. In: 17th European Conference on Machine Learning (ECML'06). LNCS, vol. 4212, pp. 282–293. Springer (2006)
7. Mehrabian, A., Anand, A., Kim, H., Sonnerat, N., Balog, M., Comanici, G., Berariu, T., Lee, A., Ruoss, A., Bulanova, A., et al.: Finding increasingly large extremal graphs with alphazero and tabu search. arXiv preprint arXiv:2311.03583 (2023)
8. Rosin, C.D.: Nested rollout policy adaptation for monte carlo tree search. In: In IJCAI. pp. 649–654 (2011)
9. Roucairol, M., Cazenave, T.: Refutation of spectral graph theory conjectures with monte carlo search. In: International Computing and Combinatorics Conference. pp. 162–176. Springer (2022)
10. Roucairol, M., Cazenave, T.: Solving the hydrophobic-polar model with nested monte carlo search. In: International Conference on Computational Collective Intelligence. pp. 619–631. Springer (2023)
11. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Driessche, G.v.d., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of Go with deep neural networks and tree search. Nature **529**, 484–489 (2016)
12. Wiggers, W., van Bergen, W.: A comparison of a genetic algorithm and a depth first search algorithm applied to japanese nonograms. In: Twente student conference on IT. Citeseer (2004)

Chapter 5

Conclusion

Four major points should be retained from this work:

- 1) The applications of Monte Carlo Tree Search algorithms are many. This class of algorithm can be applied to any problem with defined actions and evaluation functions. They are the state of the art on many combinatorics problems. While they are not apt to generate images like recent advances in deep learning, they are more efficient at solving more strictly defined problems that can lead to great results such as in pharmaceutics for example. MCTS algorithms are so versatile that all projects in this thesis use the same algorithms and only swap the search space.
- 2) UCT is by far the most commonly used algorithm, and family of algorithms. But it was shown to be outperformed by the nested family algorithms on each problem from this thesis. This is because the nested algorithms optimize at any depth of the search tree and spend less time in local maximums. Greedy algorithms also showed good performance in combinatorics, especially in mathematics. We encourage future projects to try more algorithms than just UCT.
- 3) The LNMCS, a variant of NMCS, introduced in this thesis, is the best-performing MCTS algorithm. It reintroduces the exploitation/exploration dilemma and avoid traps the NMCS could fall into.
- 4) Neural networks can be outperformed by simpler methods, especially in combinatorics and mathematical problems. Neural networks still are necessary, or very useful, as policies for complex and less combinatoric tasks such as retrosynthesis, or complex evaluation functions.

Here is an exhaustive list of what was accomplished during this thesis:

- 1a) The superiority of MCTS and sometimes greedy search algorithms over deep learning methods was shown on the spectral graph theory

refutation problem. The constructive approach of the search space is also a reason for the good performances. The nested family of algorithms produces the best results on this problem.

- 1b) Conjecture 197 from Graffiti is refuted. No refuted conjecture from Graffiti is left unrefuted by our approach. Conjectures that took days and were the subject of an article are refuted in seconds or minutes.

- 1c) Three other lesser-known automatically generated conjectures are refuted. Among dozen of them which were refuted more effectively by our approach.

- 2a) MCTS and deterministic search algorithms were showed to outperform the state of the art for the optimization of communications and transport graphs under budget constraints.

- 2b) The optimization of communications and transport graphs under budget constraints is a complex problem whose variants react differently to different algorithms.

- 2c) The optimization of communications and transport graphs under budget constraints synthetic instances share the same properties in optimization as the real-world instances despite their visible differences.

- 3a) The Coalition Structure Generation problem has a new state of the art, vastly outperforming the previous, and showing the strength of LNMCS.

- 3b) The Coalition Structure Generation problem would benefit from not having too many redundant variation which show no differences between the algorithms' performances among them and no reason for their existence.

- 4a) The HP model has the LNMCS, a new, open, and powerful algorithm for solving it. The HP model is a very interesting toy model with an unforgiving search space, we encourage researchers in optimization to try their search algorithms on it.

- 4b) The previous SOTA methods for solving the HP model were not made public, and were not given a pseudocode. Attempts to reproduce them did not yield the announced results, to the best of our knowledge, LNMCS is the best method for the HP model with a readily available code.

- 5a) Replacing AiZynthFinder's MCTS with NMCS or GBFS greatly improved the retrosynthesis' tool performances.

- 5b) Improving the neural network architecture and method also improved the performances.
- 5c) Combined together, these improvements allowed AiZynthFinder (already SOTA before the improvements) to vastly outperform ASKCOS and other SOTA methods. Despite a much smaller dataset compared to the other approaches.
- 6a) A new SOTA de novo drug generator was proposed, using ngrams instead of deep learning, and growing the molecules character by character from the SMILES representation.
- 6b) This new method is able to produce unique, novel, and valid molecules like none before. In addition, at least 32% of the molecules were synthesizable, which was not directly assessed using a retrosynthetic software before.
- 7a) An attempt was made to evaluate the difficulty of nonograms using a human-like solver, with relative and subjective success.
- 7b) An attempt was made to evaluate the fun of nonograms using a human-like solver, evaluating fun is highly subjective and a harder task that was not accomplished without flaws.

This thesis shows that search algorithms, and especially MCTS (and more precisely, the nested family of algorithms) can easily produce the best results in optimization processes in constructive search spaces.

Ce document regroupe les article publiés lors de ma thèse dirigée par Tristan Cazenave au LAMSADE.
La recherche Monte Carlo désigne une classe d'algorithmes de recherche stochastiques retournant une solution avec une garantie dans le temps, mais sans garantie de résultat. Ces algorithmes utilisent des techniques d'apprentissage par renforcement basées sur des exploration aléatoires ou guidées.
Les capacités des algorithmes Monte Carlo sont limitées dans des domaines d'application mis en valeur récement, comme la génération d'image et de texte, ou les réseaux de neurones, LLM et autres algorithmes entrainés sur de larges bases de données dominent. Mais en revanche ils excellent sur les problèmes plus classiques et définis.
L'usage le plus connu d'algorithme de recherche Monte Carlo est son utilisation en 2017 pour battre pour la première fois un champion de Go, chose qu'aucune autre famille d'algorithme n'était parvenue à faire. Mais les utilisations d'algorithmes de recherche Monte Carlo vont aussi bien au delà des jeux. Les algorithmes de recherche Monte Carlo sont largement utilisés dans la chimie, la recherche opérationelle, les transports, les mathématiques, et dans les jeux. Ils peuvent être appliqués à tout problème de décision séquentielle et de recherche dans un espace d'état tant que les fonctions d'évaluation et de modification d'un état sont définies.
La définition de structure pour cette thèse est ``système défini par les éléments qui le composent et les interactions entre ces éléments''. Cette thèse explore plusieures applications de recherche Monte Carlo dans le contexte de la génération de structure. De nombreux espaces de recherche peuvent être représentés comme une structure en dehors des jeux, comme le circuit du problème de voyageur de commerce par exemple, mais aussi des molécules, des cristaux, des coalitions, des graphes, etc.
Les points forts de cette thèse sont : - Des comparaisons entre algorithmes sur divers problèmes montrant la supériorité de la famille d'algorithmes ``nested''. - Une nouvelle variante de la Nested Monte Carlo Tree Search (NMCS) avec de meilleures performances. - Une bibliothèque d'algorithmes Monte Carlo codés en Rust. - Un projet de réfutation de conjectures des graphes. - Une implémentation du NMCS pour AiZynthFinder, le logiciel de rétrosynthèse open source d'AstraZeneca. - Un programme de génération de molécules valides et synthétisables.
Les sujets abordés peuvent être séparés en deux groupes. D'un côté la chimie, avec le HP-model, la rétrosynthèse, et la génération de molécules. Et de l'autre les mathématiques, avec les structures de coalitions, la théorie spectrale des graphes, les réseaux de transport, et les nonograms.
Bien que cette thèse ne se consacre qu'à des applications de la recherche Monte Carlo, elle apporte aussi des aperçus plus généraux : une comparaison des familles d'algorithmes montrant la supériorité des ``nested'', une nouvelle variante du NMCS, et des heuristiques et modifications généralement utiles pour les problèmes combinatoirement difficiles.

MOTS CLÉS

Monte Carlo, Optimisation, Algorithmes, Recherche, Design de Molécules, Transports, Réseaux, Théorie des Graphes, Puzzles, Coalitions

ABSTRACT

This document gathers the articles published during my PhD thesis directed by Tristan Cazenave at LAMSADE.
Monte Carlo search refers to a class of stochastic search algorithms that return a solution with a guarantee of time, but no guarantee of result. These algorithms use reinforcement learning techniques based on random or guided exploration. The capabilities of Monte Carlo algorithms are limited in recently highlighted application domains, such as image and text generation, where neural networks, LLM and other algorithms trained on large databases dominate. On the other hand, they excel in more classic, defined problems.
The best-known use of a Monte Carlo search algorithm is its use in 2017 to beat a Go champion for the first time, something no other algorithm family had managed to do. But the uses of Monte Carlo search algorithms also extend far beyond gaming. Monte Carlo search algorithms are widely used in chemistry, operations research, transportation, mathematics, and gaming. They can be applied to any sequential decision and state-space search problem, as long as the functions for evaluating and modifying a state are defined.
The definition of structure for this thesis is “a system defined by the elements that compose it and the interactions between these elements”. This thesis explores several applications of Monte Carlo search in the context of structure generation. Many search spaces can be represented as structures outside of games, such as the circuit of the traveling salesman problem, but also molecules, crystals, coalitions, graphs, etc.
The highlights of this thesis are: - Comparisons between algorithms on various problems showing the superiority of the “nested” family of algorithms. - A new variant of Nested Monte Carlo Tree Search (NMCS) with improved performance. - A library of Monte Carlo algorithms coded in Rust. - A project to refute graph conjectures. - An NMCS implementation for AiZynthFinder, AstraZeneca's open source retrosynthesis software. - A program for generating valid, synthesizable molecules.
The topics covered can be divided into two groups. On the one hand, chemistry, with HP-model, retrosynthesis and molecule generation. On the other, mathematics, with coalition structures, spectral graph theory, transport networks and nonograms.
Although this thesis is devoted solely to applications of Monte Carlo search, it also provides more general insights: a comparison of algorithm families showing the superiority of “nested”, a new variant of NMCS, and heuristics and modifications generally useful with NP problems.

KEYWORDS

Monte Carlo, Optimization, Algorithms, Research, De Novo Drug Design, Transportations, Networks, Graph Theory, Puzzles, Coalitions