Monte Carlo Search Algorithms Discovering Monte Carlo Tree Search Exploration Terms

Tristan Cazenave

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

Abstract. Monte Carlo Tree Search and Monte Carlo Search have good results for many combinatorial problems. In this paper we propose to use Monte Carlo Search to design mathematical expressions that are used as exploration terms for Monte Carlo Tree Search algorithms. The Monte Carlo Tree Search algorithm we aim to optimize is SHUSS (Sequential Halving Using Scores). We automatically design the SHUSS root exploration term. For small search budgets of 32 evaluations the discovered root exploration term makes SHUSS competitive with usual PUCT (Predictor Upper Confidence bounds for Trees).

1 Introduction

Monte Carlo Tree Search [13, 23] is a well known family of algorithms that were designed for the game of Go and then applied to many different combinatorial problems [2, 42].

Our goal in this paper is to use Monte Carlo Search to improve Monte Carlo Tree Search. This is part of a longstanding goal of using Artificial Intelligence to improve Artificial Intelligence [33].

The usual way to design an exploration term is to make a theoretical analysis [1]. We take another empirical approach. We randomly generate many exploration terms and keep the ones that work well in practice. This is a simpler approach, yet it can find exploration terms that work well in practice and that surpass the ones found with a theoretical analysis.

Another approach to the automatic improvement of Monte Carlo Tree Search exploration terms is to use Genetic Programming. It could evolve Monte Carlo Tree Search algorithms, improving on UCT and RAVE for the game of Go [5]. However this approach relies on making the exploration terms play against each other which is time consuming. It is also more complicated than the method we propose in this paper.

Monte Carlo Tree Search combined with Deep Reinforcement Learning has been used to improve algorithms. AlphaTensor discovered new fast matrix multiplications algorithms playing the tensor game [16]. AlphaTensor as well as other Monte Carlo Tree Search algorithms have also been used for quantum circuit optimization [20, 43, 34, 35]. New fast sorting algorithms were discovered thanks to Monte Carlo Tree Search with the AlphaDev system [27].

Monte Carlo Search has been used for discovering mathematical expressions that maximize a given score function [6, 7]. This was applied to different domains including physics [41], finance [10], and the automated design of functions [21].

Refinements of the Monte Carlo Search approach to mathematical expressions discovery include incorporating actor-critic in Monte Carlo Tree Search for symbolic regression [25], using a grammar of Monte Carlo Search algorithms [26], controlling the size [29], and using GPT as a prior [24].

The automated discovery of optimization algorithms with symbolic program search recently enabled to discover a simple and effective optimization algorithm, Lion (evoLved sIgn mOmeNtum). Lion is more memory-efficient than Adam as it only keeps track of the momentum [12].

Our work is in line with these uses of Artificial Intelligence to discover new Artificial Intelligence algorithms. Our goal is to use Monte Carlo Search to discover a new root exploration term for the SHUSS algorithm [15].

Our contributions are:

- An efficient method to empirically design exploration terms.
- The AMAF prior for non uniform playouts in Monte Carlo Search applied to the discovery of mathematical expressions.
- The design of a curriculum learning dataset for discovering exploration terms.
- A better way of selecting moves for SHUSS according to their priors given by the policy network.

The second section presents various Monte Carlo Tree Search algorithms. The third section explains how we generate mathematical expressions for the exploration terms. The fourth section details experimental results.

2 Monte Carlo Tree Search

In this section we present various Monte Carlo Tree Search algorithms, starting with PUCT the most popular one which is used in Alpha Zero and that is standard in computer games. We then define the AMAF prior that can be used to play non uniform playouts biased toward the actions that give better playouts scores. It uses statistics on the playouts that contain an action to calculate its probability of being played as explained in the following subsection on sampling. We then present the Sequential Halving algorithm as well as the related Sequential Halving Using Scores (SHUSS) algorithm. We end this section explaining how generated exploration terms can be used for SHUSS.

2.1 PUCT

Monte Carlo Tree Search was designed for computer Go and made a revolution in computer Go [13, 23] and then in computer game playing [17, 28]. The current standard algorithm for Monte Carlo Tree Search is PUCT. This is the search algorithm used in AlphaGo [38], AlphaGo Zero [39], AlphaZero [40] and MuZero [37].

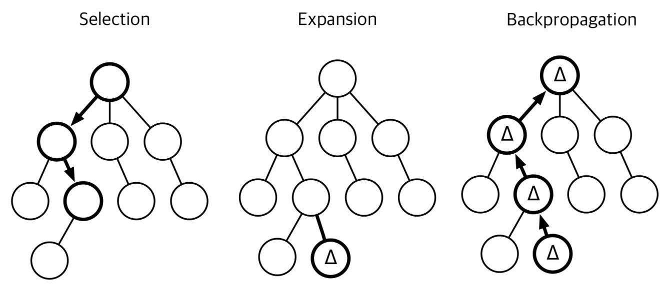

Fig. 1: The three steps of MCTS. The first step is the tree descent using the exploration term to choose among the children. The second step is adding a new leaf associated to an evaluation of the state by the value network. The third step is updating the statistics in the tree with the evaluation.

Figure 1 gives the three steps of MCTS. The principle of the algorithm is to memorize the already explored states as well as the associated statistics for the possible actions in these states. When the algorithm encounters a state he has already explored, it uses an exploration term to choose the next action to play. The average of the previous evaluations associated to an action a in state s is $Q(s,a)$. The number of descents that have passed by state s is $N(s)$ and the number of descent in s that have played action a is $N(s,a)$.

A neural network is used at the leaves of the tree to evaluate the leaf and calculate probabilities for the possible actions. The probability, as estimated by the neural network, that action a is the best in state s is $P(s,a)$.

The exploration term which is added to $Q(s,a)$ in PUCT is:

$$c_p \times P(s,a) \times \frac{\sqrt{N(s)}}{1+N(s,a)}$$

The constant c_p is an hyper parameter that has to be tuned for each problem.

In the following we use pr for $P(s,a)$ the probability for action a in state s given by the neural network policy head.

2.2 The AMAF prior

The All Moves As First (AMAF) heuristic comes from computer Go [4]. It calculates statistics on moves independently of when the moves were played in a playout. It is used in General Game Playing [32] in the MAST algorithm [18]. It was also used in computer Go in the RAVE [19] algorithm. The principle of RAVE is to bias the tree policy with the AMAF statistics of the node. RAVE is much better than UCT for the game of Go. RAVE was later generalized to GRAVE [8] by using AMAF statistics of an ancestor node of the tree instead of the AMAF statistics of the node. GRAVE has much better results than RAVE for many games. It has good results in General Game Playing. It is the standard Monte Carlo Tree Search algorithm used in the Ludii system [3].

We now define a more elaborate version of AMAF. If $\mathcal{P}$ is the set of the playouts and $s(p)$ is the score of playout $p \in \mathcal{P}$, we define:

$$\mu = \frac{\sum_{p \in \mathcal{P}} s(p)}{|\mathcal{P}|}$$

$$\mathcal{P}_a = \{p \in \mathcal{P} \mid a \in p\}$$

$$\mu_a = \frac{\sum_{p \in \mathcal{P}_a} s(p)}{|\mathcal{P}_a|} - \mu$$

$$maxi = \max_a(|\mu_a|)$$

$$z = \sum_a e^{\frac{\mu_a}{maxi}}$$

$$AMAF(a) = \frac{e^{\frac{\mu_a}{maxi}}}{z}$$

2.3 Sampling

The basic algorithm in Monte Carlo Search is sampling. It performs playouts by randomly choosing actions until it reaches a terminal state.

It usually improves the results of the playout to adopt a non uniform strategy for sampling. A policy can attribute different probabilities to the possible actions in a state. The sampling algorithm can then choose the next action to play according to these probabilities.

It is also possible to use a temperature τ to make the policy more or less exploratory. In the case of AMAF, if $\mathcal{A}$ is the set of the possible actions in state s, sampling with a temperature τ consists in choosing the next action a with probability p_a:

$$p_a = \frac{e^{\frac{log(AMAF(a))}{\tau}}}{z}$$

$$z = \sum_{a \in \mathcal{A}} e^{\frac{log(AMAF(a))}{\tau}}$$

2.4 Sequential Halving

Sequential Halving [22] is an algorithm that minimizes the simple regret. It has successfully been used as an alternative to UCB in Monte Carlo Tree Search, in particular as a replacement in the root node with UCB used in the rest of the tree [31], or even in the whole tree with SHOT [9]. It was applied to games as well as to partially observable games [30]. Sequential Halving was also used as a root policy with Gumbel MuZero [14]. The outputs of the Sequential Halving at the root were used for reinforcement learning in the MuZero algorithm. Gumbel MuZero was successfully applied to Go, Chess and Atari games.

Algorithm 1 gives the Sequential Halving algorithm used in SHUSS [15]. This is the one we use in this paper with $\lambda = \frac{1}{2}$. The principle of the algorithm is to allocate the same number of playouts to all the actions in the set of actions S_r. It then selects half of the actions in S_r that have the best empirical average. This best half constitutes S_{r+1}. The algorithm continues to allocate playouts to remaining actions and to select the best half until there is only one action remaining in S_R.

Algorithm 1 Sequential Halving

Parameter: cutting ratio λ
Input: total budget T, set of arms S
$S_0 \leftarrow S$, $T_0 \leftarrow T$
$R \leftarrow$ number of rounds before $|S_R| = 1$
for $r = 0$ **to** $R - 1$ **do**
  $t_r \leftarrow \lfloor \frac{T_r}{|S_r| \cdot (R-r)} \rfloor$
  $T_{r+1} \leftarrow T_r - t_r |S_r|$
  sample t_r times each arm in S_r
  $S_{r+1} \leftarrow S_r$ deprived of the fraction $1 - \lambda$ of the worst arms
end for
Output: arm in S_R

2.5 Sequential Halving Using Scores

SHUSS [15] is an improvement of Sequential Halving that uses a prior to improve the move selection at the root. The prior can be used either to eliminate moves or to bias the selection of the actions.

When the prior is standard AMAF it selects the moves to keep using:

$$\tilde{Q}_a = Q_a + C \times \frac{StandardAMAF(a)}{N(root, a)}$$

$$StandardAMAF(a) = \frac{\sum_{p \in \mathcal{P}_a} s(p)}{|\mathcal{P}_a|}$$

In case of a prior given by a neural network, it uses the classic Sequential Halving algorithm restricted to a fixed number of moves that have the best priors.

2.6 Using an exploration term for the selection of moves in SHUSS

In the same spirit as SHUSS it is possible to use various exploration terms for choosing the moves to keep at the end of a round of Sequential Halving. Algorithm 2 gives the move selection process with an exploration term. The principle is to take the best half of the moves that maximize the expression. If the expression is the usual empirical average then the algorithm is the usual Sequential Halving algorithm.

Algorithm 2 Selection of the moves to keep for the next round

Parameter: cutting ratio λ
Input: set of moves S_r
$S_{r+1} \leftarrow \emptyset$
for $i = 0$ **to** $\lambda \times |S_r|$ **do**
  $bestScore \leftarrow -\infty$
  for $j = 0$ **to** $|S_r|$ **do**
    if $S_r[j] \notin S_{r+1}$ **then**
      if $expression(S_r[j]) > bestScore$ **then**
        $bestMove \leftarrow S_r[j]$
        $bestScore \leftarrow expression(S_r[j])$
      end if
    end if
  end for
  $S_{r+1} \leftarrow S_{r+1} \cup \{bestMove\}$
end for
return S_{r+1}

3 Generating Mathematical Expressions

In this section we detail the algorithms we use to discover mathematical expressions. We first define the expression discovery game, and then explain how to sample expressions for this game.

3.1 The expression discovery game

The expression discovery game is used to generate and evaluate expressions that are used as the SHUSS exploration term. Expression trees are represented as stacks in reverse polish notation. For example the generated expression [+, pr, *, *, 2, sc, sc] corresponds to the exploration term $pr + 2 \times sc \times sc$ where sc is the sum of the scores of the playouts starting with the move to evaluate and *pr* the prior for that move. The evaluation of an expression in reverse polish notation is algorithmically simple as it uses a straightforward depth first search.

In order to limit the size of the generated expressions, we maintain the number of open leaves of an incomplete expression. This is the total number of children of the atoms of the expression that are not yet associated to an atom. In the root node the number of open leaves of the empty expression is 1. If for example a '+' atom is assigned to the root, then there is one open leaf less due to the assignment and two open leaves more due to the '+' having two children not yet assigned. In order to generate expressions that are smaller than the maximum length, the legal moves function does not return atoms that make the number of already assigned atoms plus the number of open leaves plus the number of children of the atom greater than the maximum length.

The atoms we used to generate expressions are:

- 1, 2, 3 and 100 numbers.
- sc: the sum of the scores of the playouts starting with the move.
- pr: the prior for the move given by the policy head.
- nbp: the number of playouts starting with the move.
- nb: the total number of playouts.
- +, -, *, /, log, exp, =, max and min operators.

3.2 Sampling

The default sampling procedure is uniform sampling. It is possible to replace it with non uniform sampling using the AMAF prior.

In our code for sampling mathematical expressions, the possible atoms are defined in a list and the number of children of each atom is defined in the corresponding children list. A state is a possibly incomplete mathematical expression in reverse polish notation. It is associated to a number of open leaves which is the minimum number of atoms that have to be added to the expression in order to have a complete expression.

The usual functions to define a problem for Monte Carlo Search are defined as follows for the mathematical expression discovery game:

- The legal moves function takes as parameters an incomplete expression and the number of associated open leaves. It returns the list of atoms that can be added to the incomplete expression. It verifies that adding an atom does not exceed the maximal number of atoms for the final complete expression.
- The play function just adds the selected atom to the expression and also returns the updated number of open leaves.
- The terminal function returns True when the expression is complete.
- The playout function is the usual uniformly random playout function that randomly adds authorized atoms to the expression until the expression is complete.

4 Experimental Results

In this section we experiment the discovery of Monte Carlo Tree Search root exploration terms in the game of Go. We present the computer Go dataset that was used to train a transformer network for the game of Go. The transformer network is used to generate the SHUSS dataset that is in turn used to evaluate generated root exploration terms. We compare uniform sampling to AMAF sampling for the discovery of root exploration terms. We apply the framework to the discovery of root exploration terms for SHUSS. We then test the discovered exploration terms in a Go program, making the new algorithm play against standard PUCT.

4.1 The computer Go dataset

The computer Go dataset is composed of games played by Katago [44] against itself in 2022. There are 1,000,000 different games in total in the training set. The input data is composed of 31 19x19 planes (color to play, ladders, current state on two planes, two previous states on four planes). The output targets are the policy (a vector of size 361 with 1.0 for the move played, 0.0 for the other moves), and the value (close to 1.0 if White wins, close to 0.0 if Black wins). The test set is composed of 50,000 states taken randomly from 50,000 games that are not used in the training set.

4.2 The neural network

We trained a computer Go vision transformer network [36] using 2,000 epochs with 100,000 states per epoch. The loss for the policy head is a categorical cross entropy and the loss for the value head is a binary cross entropy. The optimizer is Adam and the learning rate decreases according to a cosine annealing [11]. The network reached an accuracy of 57.75% on Katago moves, a Mean Squared Error (MSE) of 0.0334 and a Mean Absolute Error (MAE) of 0.117.

4.3 The SHUSS dataset

The time required to precisely evaluate a generated expression in a real Go playing program can be huge. For example making a Sequential Halving algorithm with a generated expression play against a standard PUCT for 500 games with 1024 playouts per move takes days.

In order to have a fast evaluation of a given exploration term we built a dataset of states associated to their cached search. The policy learned by the neural network is of high quality. Out of the 2,000 states taken from the test set only 77 of them have a prior less than 0.01. In the remaining of the experiments, we only use moves that have a prior greater or equal to 0.01 for Sequential Halving. For each of the moves that have a prior greater or equal to 0.01, we call PUCT starting with the move a fixed number of times (e.g. 32) and store the sequence of evaluations returned by the successive calls to PUCT after the fixed first move. Therefore for all moves we have an associated sequence of 32 evaluations that can be used to simulate the calls made by Sequential Halving while not using the inference by the neural network. This enables a very fast evaluation of a given Sequential Halving exploration term as the neural network is not used anymore to calculate the accuracy of the exploration term. The label of a state is the move found by Sequential Halving with 128 evaluations. The accuracy of an exploration term is defined as the percentage of labeled moves found by the Sequential Halving algorithm with 32 evaluations on the 2,000 cached states.

In order to speed up the evaluation of the exploration terms we stop evaluating the accuracy of an exploration term if it scores less than a threshold of 80 after 200 states. We also use memoization of the scores of the exploration terms in a dictionary as well as the memoization of the sums of scores for a given number of playouts and a given first move.

In the experiments we run 100 processes generating exploration terms in parallel for 512 seconds. It results in 354,400 exploration terms being evaluated.

4.4 AMAF sampling

The evolution of the best accuracy on the Sequential Halving moves with 128 evaluations is given in Figure 2 both for uniform sampling and for AMAF sampling. AMAF sampling is much better.

4.5 Discovering a SHUSS exploration term

Table 1 gives the accuracy of different exploration terms for the Sequential Halving moves. The prior pr is much worse than the standard Sequential Halving algorithm sc on this dataset. The sampling algorithm finds the $sc \times (pr + sc \times sc)$ exploration term that is better than both.

Exploration Term	Accuracy on the SHUSS dataset
pr	44.85%
sc	71.45%
$pr + 2 \times sc \times sc$	72.85%

Table 1: Accuracy of SHUSS with 32 evaluations on the SHUSS dataset. The SHUSS label moves are found using Sequential Halving with 128 evaluations. The accuracy is calculated on the moves found by SHUSS with 32 evaluations and the depicted exploration term for halving. The sc exploration term corresponds to standard SHUSS. The $pr + 2 \times sc \times sc$ has a slightly better accuracy on the SHUSS dataset.

4.6 Testing the SHUSS exploration term in a Go program

We evaluate the different exploration terms by using them in a computer Go program. The SHUSS algorithm using an exploration term plays 400 games against PUCT. Both algorithms use 32 evaluations for choosing their move. PUCT chooses the most simulated move and SHUSS chooses the only remaining move in S_R. The results for standard SHUSS with the usual exploration term sc are given in Table 2. We can observe that the standard SHUSS is weaker than PUCT. The exploration term $pr + 2 \times sc \times sc$ is also tested against PUCT. For all the PUCT constants we tested, SHUSS with the discovered exploration term is better than PUCT. The best result for PUCT is that SHUSS wins 51.00% of its games. So we can say that the discovered exploration term made SHUSS competitive with PUCT.

If we analyze the discovered exploration term $pr + 2 \times sc \times sc$, we see that when there are only a few playouts it takes into account the prior so as not to eliminate moves that have a great prior. When the number of playouts is greater, the $2 \times sc \times sc$ value becomes much greater than the prior and the moves are sorted according to the square of the sum of their scores and not much according to their prior anymore.

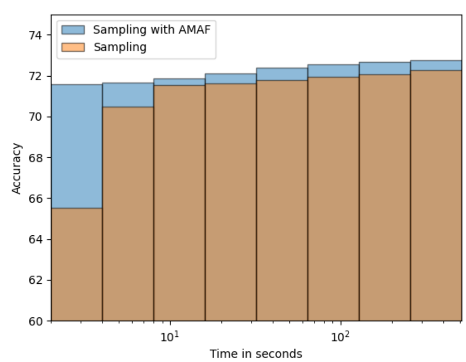

Fig. 2: Evolution of the best expression accuracy with the logarithm of the sampling search time with doubling search times. Each measure is the average of 100 runs of the sampling algorithms. Using the AMAF prior improves the results. It finds the same accuracy more than 8 times faster than the uniform sampling algorithm. The temperature of the AMAF sampling is set to $\frac{1}{5}$. The dataset used is the Sequential Halving moves with 128 evaluations dataset and the exploration terms are scored using 32 evaluations on each state out of the 2,000 states.

Exploration Term	Winrate against PUCT
sc	42.50%
$pr + 2 \times sc \times sc$	51.00%

Table 2: Winrates of standard SHUSS (the exploration term is sc) and SHUSS with the $pr + 2 \times sc \times sc$ exploration term against PUCT. The two SHUSS algorithms use 32 evaluations and the 5 best prior moves. PUCT also uses 32 evaluations. We see that the discovered exploration term is an improvement on standard SHUSS. The 400 starting states are taken from the Katago dataset test set games by playing 20 moves by Katago from the beginning of the games. The resulting states are balanced according to Katago.

5 Conclusion

We presented a simple yet efficient method to find new exploration terms for Monte Carlo Tree Search. It uses sampling of mathematical expressions and a fast evaluation of the generated expressions. The generated exploration terms are simple. For search with a small number of evaluations, the method discovered an exploration term that works better than the usual exploration term for Sequential Halving. The discovered exploration term also beats the canonical PUCT algorithm for small equivalent search times. We also proposed the AMAF prior for sampling mathematical expressions. It reaches a score approximately 8 times faster than uniform sampling.

Our method to discover new exploration terms is simple, fast, general and empirically adapts the generated mathematical expressions to the problem at hand.

Future work involves accelerating the discovery of the expressions and applying the algorithm to other problems. It would also be interesting investigating the generation of more general expressions by evaluating them on more varied data, for example with different numbers of playouts or even for different games or problems.

From a more general point of view, Artificial Intelligence is becoming powerful enough to help discover new Artificial Intelligence algorithms. There are many further developments along the line of using Artificial Intelligence to improve Artificial Intelligence.

References

1. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Mach. Learn. **47**(2-3), 235–256 (2002)
2. 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 TCIAIG **4**(1), 1–43 (Mar 2012)

3. Browne, C., Stephenson, M., Piette, É., Soemers, D.J.: A practical introduction to the ludii general game system. In: ACG 2019. pp. 167–179. Springer (2020)
4. Brügmann, B.: Monte Carlo Go. Tech. rep., Max-Planke-Inst. Phys., Munich (1993)
5. Cazenave, T.: Evolving monte carlo tree search algorithms. Dept. Inf., Univ. Paris **8** (2007)
6. Cazenave, T.: Nested monte-carlo expression discovery. In: ECAI 2010, pp. 1057–1058. IOS Press (2010)
7. Cazenave, T.: Monte-carlo expression discovery. International Journal on Artificial Intelligence Tools **22**(01), 1250035 (2013)
8. Cazenave, T.: Generalized rapid action value estimation. In: Yang, Q., Wooldridge, M.J. (eds.) IJCAI 2015. pp. 754–760. AAAI Press (2015)
9. Cazenave, T.: Sequential halving applied to trees. IEEE Trans. Comput. Intell. AI Games **7**(1), 102–105 (2015)
10. Cazenave, T., Hamida, S.B.: Forecasting financial volatility using nested monte carlo expression discovery. In: IEEE SSCI. pp. 726–733 (2015)
11. Cazenave, T., Sentuc, J., Videau, M.: Cosine annealing, mixnet and swish activation for computer Go. In: Advances in Computer Games. Springer (2021)
12. Chen, X., Liang, C., Huang, D., Real, E., Wang, K., Pham, H., Dong, X., Luong, T., Hsieh, C.J., Lu, Y., et al.: Symbolic discovery of optimization algorithms. Advances in Neural Information Processing Systems **36** (2024)
13. Coulom, R.: Efficient selectivity and backup operators in Monte-Carlo tree search. In: van den Herik, H.J., Ciancarini, P., Donkers, H.H.L.M. (eds.) CG 2006. Lecture Notes in Computer Science, vol. 4630, pp. 72–83. Springer (2006)
14. Danihelka, I., Guez, A., Schrittwieser, J., Silver, D.: Policy improvement by planning with gumbel. In: International Conference on Learning Representations (2021)
15. Fabiano, N., Cazenave, T.: Sequential halving using scores. In: Advances in Computer Games, pp. 41–52. Springer (2021)
16. Fawzi, A., Balog, M., Huang, A., Hubert, T., Romera-Paredes, B., Barekatain, M., Novikov, A., R Ruiz, F.J., Schrittwieser, J., Swirszcz, G., et al.: Discovering faster matrix multiplication algorithms with reinforcement learning. Nature **610**(7930), 47–53 (2022)
17. Finnsson, H., Björnsson, Y.: Simulation-based approach to general game playing. In: AAAI. pp. 259–264 (2008)
18. Finnsson, H., Björnsson, Y.: Learning simulation control in general game-playing agents. In: AAAI (2010)
19. Gelly, S., Silver, D.: Monte-Carlo tree search and rapid action value estimation in computer Go. Artif. Intell. **175**(11), 1856–1875 (2011)
20. Hummel, A., Cazenave, T.: Monte carlo qubit routing. In: Quantum Machine Learning at ECML PKDD (2022)
21. Illetskova, M., Elnabarawy, I., Da Silva, L.E.B., Tauritz, D.R., Wunsch, D.C.: Nested monte carlo search expression discovery for the automated design of fuzzy ART category choice functions. In: GECCO. pp. 171–172 (2019)
22. Karnin, Z.S., Koren, T., Somekh, O.: Almost optimal exploration in multi-armed bandits. In: ICML. pp. 1238–1246 (2013)
23. 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)
24. Li, Y., Li, W., Yu, L., Wu, M., Liu, J., Li, W., Hao, M., Wei, S., Deng, Y.: Discovering mathematical formulas from data via GPT-guided monte carlo tree search. arXiv preprint arXiv:2401.14424 (2024)

25. Lu, Q., Tao, F., Zhou, S., Wang, Z.: Incorporating actor-critic in monte carlo tree search for symbolic regression. Neural Computing and Applications **33**, 8495–8511 (2021)
26. Maes, F., St-Pierre, D.L., Ernst, D.: Monte carlo search algorithm discovery for single-player games. IEEE Transactions on Computational Intelligence and AI in Games **5**(3), 201–213 (2013)
27. Mankowitz, D.J., Michi, A., Zhernov, A., Gelmi, M., Selvi, M., Paduraru, C., Leurent, E., Iqbal, S., Lespiau, J.B., Ahern, A., et al.: Faster sorting algorithms discovered using deep reinforcement learning. Nature **618**(7964), 257–263 (2023)
28. Méhat, J., Cazenave, T.: Monte-Carlo Tree Search for general game playing. Univ. Paris **8** (2008)
29. Moudřík, J., Křen, T., Neruda, R.: Algorithm discovery with monte-carlo search: Controlling the size. In: ICTAI. pp. 390–395. IEEE (2017)
30. Pepels, T., Cazenave, T., Winands, M.H.M.: Sequential halving for partially observable games. In: Cazenave, T., Winands, M.H.M., Edelkamp, S., Schiffel, S., Thielscher, M., Togelius, J. (eds.) CGW 2015. Communications in Computer and Information Science, vol. 614, pp. 16–29. Springer (2015)
31. Pepels, T., Cazenave, T., Winands, M.H.M., Lanctot, M.: Minimizing simple and cumulative regret in monte-carlo tree search. In: Cazenave, T., Winands, M.H.M., Björnsson, Y. (eds.) CGW 2014. Communications in Computer and Information Science, vol. 504, pp. 1–15. Springer (2014). https://doi.org/10.1007/978-3-319-14923-3_1, https://doi.org/10.1007/978-3-319-14923-3_1
32. Pitrat, J.: Realization of a general game-playing program. In: IFIP Congress (2). pp. 1570–1574 (1968)
33. Pitrat, J.: A step toward an artificial artificial intelligence scientist (2008), LIP6 Research Report
34. Rosenhahn, B., Osborne, T.J.: Monte carlo graph search for quantum circuit optimization. Physical Review A **108**(6), 062615 (2023)
35. Ruiz, F.J., Laakkonen, T., Bausch, J., Balog, M., Barekatain, M., Heras, F.J., Novikov, A., Fitzpatrick, N., Romera-Paredes, B., van de Wetering, J., et al.: Quantum circuit optimization with alphatensor. arXiv preprint arXiv:2402.14396 (2024)
36. Sagri, A., Arjonilla, J., Saffidine, A., Cazenave, T.: Vision transformers for computer Go. In: EvoApps. Springer (2024)
37. Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al.: Mastering atari, go, chess and shogi by planning with a learned model. Nature **588**(7839), 604–609 (2020)
38. 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)
39. 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., Hassabis, D.: Mastering chess and shogi by self-play with a general reinforcement learning algorithm. CoRR **abs/1712.01815** (2017), http://arxiv.org/abs/1712.01815
40. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al.: A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science **362**(6419), 1140–1144 (2018)

41. Sun, F., Liu, Y., Wang, J.X., Sun, H.: Symbolic physics learner: Discovering governing equations via monte carlo tree search. arXiv preprint arXiv:2205.13134 (2022)
42. Świechowski, M., Godlewski, K., Sawicki, B., Mańdziuk, J.: Monte carlo tree search: A review of recent modifications and applications. Artificial Intelligence Review **56**(3), 2497–2562 (2023)
43. Wang, P., Usman, M., Parampalli, U., Hollenberg, L.C., Myers, C.R.: Automated quantum circuit design with nested monte carlo tree search. IEEE Transactions on Quantum Engineering (2023)
44. Wu, D.J.: Accelerating self-play learning in go. arXiv preprint arXiv:1902.10565 (2019)