A Survey in Algorithms: Monte-Carlo Tree Search, and it's implementations in the board games.

Ulvi Bajarani, Student ID 20539914

e-mail: ulvi.bajarani01@utrgv.edu
Class: CS 6174 Fall 2019

November 27, 2019

Abstract

The combination of the Monte Carlo method with Tree Search revealed new possibilities to grow the programs' strength in various board games. The first two chapters of the survey cover the basics of the Monte Carlo Tree Search (MCTS) and a brief definition of the MCTS enhancements. The most significant part of the work is dedicated to the MCTS implementations in various board games. Not explaining the rules of games, the author points out the circumstances and the results of experiments with the MCTS and its enhancements against various search algorithms, such as the default MCTS, the alpha-beta pruning, and describes the reason of some failures in some games and possible improvements of conditions. The final chapter concludes the survey, giving the relationship between the type of games and the efficiency of MCTS.

Keywords

Monte-Carlo, Monte-Carlo Tree Search (MCTS), Selection, Expansion, Simulation, Backpropagation, alpha-beta pruning (alpha-beta search), Upper Confidence bound algorithm applied to Trees (UCT), Rapid Action Value Estimate (RAVE), Progressive Bias, Progressive Unpruning and Progressive Widening, Heavy Playouts, Decisive Moves, Monte-Carlo Solver, Go, Havannah, Hex, Lines of Actions (LOA), Amazons, Chess, Shogi.

1 Introduction

Despite that Monte Carlo method, which based on random simulations, was discovered in the 1940s, only after the Ph.D. Bruce Abramson's 1987-year thesis work “The Expected-Outcome Model of Two-Player Games” described the possibility of using the method on board games instead of static evaluation methods, which used alpha-beta search [1]. After that, it was firstly implemented in 1992 by Brügmann [2] in a board-game program for playing Go. Since then, there have been developed several realizations and improvements of the method; the most significant was combining the method with Upper Confidence bound algorithm applied to Trees (UCT), which named as Monte-Carlo Tree Search (MCTS) and described a tremendous improvement and a breakthrough in board games, such as Go, the games of Amazons, chess, Shogi etc what is a subject of the paper.

1.1 Monte-Carlo Tree Search

MCTS analyzes most prospective moves and expands the search tree, which is based on a random sampling of the extant search tree. The expansion occurs through many playouts, also called roll-outs. These playouts continue until the end; the moves of games are selected at random. The final result of the games is used to update the weight of game tree nodes: the better the result of the node is, the higher the probability that the move is chosen in future playouts. In Pure Monte Carlo Game Search, the number of initial playouts is limited to a number. However, the efficiency of search rises with time, as more playouts are assigned to the moves with a high winning frequency according to the results of previous playouts. Monte-Carlo Tree Search consists of four sequential stages:

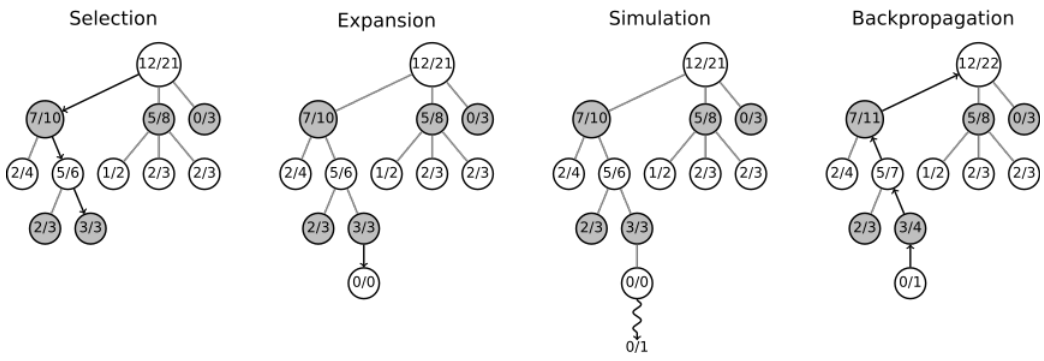
1) Selection – Starting from a root R, which is a game state, the most successive nodes are traversed till leaf L, which is the move that has not been simulated.

2) Expansion – Until a leaf L ends with a possible result, child nodes of C are created. Then, one of the child nodes is chosen until a possible result is found.

3) Simulation – a random playout of child node C.

4) Backpropagation – propagate the results through the tree from C to L and R and update their information.

5) Final Decision – After several iterations of steps from 1 to 4, the final move is selected.



1.2 Markov Decision Processes

How might a powerful method such as the Monte-Carlo Tree Search be modeled? The decision problems are frequently modeled in Markov Decision Processes, which is a composition of

- States $s \in S$, where s is a general state of the system, and $S_t \in S$ is a particular state of the system at time t.
- Actions $a \in A$, where a is a general action of the system and $A_t \in A(S_t)$ is a particular action of the system at time t, which is chosen from available actions in S_t. In the case where there are no moves in a state, then the state is called terminal, which is possible in endgame positions.
- Transition probabilities $p(s^{'}|s,a)$ is the probability of moving to state $s^{'}$ by choosing state s from action a.
- Rewards $p(s,a,s^{'})$ is the value of expected reward after choosing action a in state s and moving to the new state R_{t+1}, which is equal to $r(S_t, A_t, S_{t+1})$.
- The reward discount rate $\gamma \in [0;1]$ is the parameter that reduces the importance of later-received rewards.

To make a choice between actions, the policy $\pi(a \mid s)$ is used, which defines the probability of selecting action a in the state s. In order to solve the MDP tasks, which might be either episodic or continuous, the value of return S_t , which is also called a cumulative discounted reward, should be maximized by either

$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ \ldots$

or

$G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ \ldots \gamma^k R_T$

for episodic tasks, where $T - t$ is the remaining duration of the episode.

1.3 Upper Confidence bound algorithm applied to Trees (UCT)

Another problem is the behavior of MCTS: Should the tree-search decide whether the move is worth analyzing deeply (exploitation) or find other possible alternatives and analyze them (an exploration)? The problem is similar to the multi-armed bandit problem, which also requires a decision with limited resources. There are several possible solutions to optimize such processes in MCTS, but the most frequently used approach is the Upper Confidence bound algorithm applied to Trees (UCT). There is a possible formula of UCT, which maximizes the score of s_1:

$$z = s_1 \leftarrow \operatorname*{argmax}_{j \in C_{s_1}} [\frac{w_j}{n_j} + K \sqrt{\frac{\ln(C_{s_1})}{n_{j_1}}}] \tag{1}$$

Where s_1 is the score of s_1, C_{s_1} is the child nodes of s_1, w_j is the number of wins for the j node, n_j is the number of playouts for the node j, n_{s_1} is the number of playouts for node s_1, and K is an exploration parameter to keep the balance between exploitation and exploration.

It should be noticed that the efficiency of choice depends on not only the number of simulations, but also the time consumed on stages: when there are many games in a node, it is accurate to spend much more time on the selection, however, the simulation strategy gives a higher accuracy when the number of games in node is few. [3]

2 Enhancements of MCTS

There are several enhancements to MCTS. In section 2, the main ones are described.

2.1 Rapid Action Value Estimate (RAVE)

Suppose that we have m_i moves leading from s to $s_i^{'}$. In the MCTS algorithm, only the number of playouts n_s and winning playouts w_s are stored. While in RAVE, the number and winning playouts from s to $s_i^{'}$ are stored not only in s but also in m_i moves as $n^{'}_{s,s_i^{'}}$ and $w^{'}_{s,s_i^{'}}$, respectively. This might significantly increase the bias of selection by adding a RAVE Score in the formula 1:

$$z = s_1 \leftarrow \operatorname*{argmax}_{j \in C_{s_1}} [(1-\beta)\ \frac{w_j}{n_j} + \beta\ \frac{w^{'}_{s_{1,j}}}{n^{'}_{s_{1,j}}} +\ K \sqrt{\frac{\ln(C_{s_1})}{n_{j_1}}} \tag{2}$$

, where β is a parameter approaching to 0 when n_j tends to infinity.

The possible enhancements and diversities of RAVE are:

- PoolRave biasing moves in the simulation step;
- The Last-Good-Reply (LGR), which stores the last best reply to a previous move and tests it in the playouts of other moves;
- Move-Average Sampling Technique (MAST) which is similar to PoolRave but stores the evaluation of all moves globally;
- N-gram Average Sampling Technique, which is similar to LGR, but stores the move with the highest winning percentage;

2.2 Progressive Bias

The approach here is similar to RAVE, however, instead of using the statistics of MCTS simulation, a heuristic function with progressively decaying influence is used:

$$z = s_1 \leftarrow \underset{j \in C_{s_1}}{\operatorname{argmax}}[(1-\beta)\,\frac{w_j}{n_j} + \beta\,\frac{w'_{s_{1,j}}}{n'_{s_{1,j}}} + \; K\,\sqrt{\frac{\ln(C_{s_1})}{n_{j_1}}} + f(n_{j_1})$$

, where $f(n_{j_1})$ is a function with an heuristic evaluation of child node, which might be described as $\frac{k_{bias}\,*\,H(n_i)}{n_i}$, where k_{bias} is a suitable constant for an heuristic function, and $H(n_i)$ is an heuristic function.

2.3 Progressive Unpruning and Progressive Widening

Progressive Unpruning [4] and Progressive Widening [5] are similar approaches to optimize an expansion stage. After the first evaluation of nodes during exploration, nodes with poor heuristic scores are going to be pruned till the time when the node is better exploited; then, the nodes are going to be "unpruned." The only difference is in the realizations: While Progressive Unpruning doesn't explore a small number of child nodes, Progressive Widening prunes child nodes only after some number of simulations.

2.4 Decisive Moves

Another possible enhancement is spending more time to analyze moves, which might lead to a decisive result. If it is proved that the move leads to a decisive result, it is going to be played without any consideration of other moves' evaluations, which performed better results against random searches, even though it may have taken more time [6]. Moreover, the moves preventing decisive moves might also be implemented, which is the subject of further improvements.

2.5 Heavy Playouts

The simulation strategy, where the number and speed of simulations are decreased to improve their accuracy. In this case, domain knowledge are used, which might be either implemented patterns [7], or some heuristic function [8].

2.6 Monte-Carlo Solver

Another approach similar to Decisive Moves is the Monte-Carlo Solver. The solver proves the game-theoretical value of a node when it is possible. Monte-Carlo Solver proves only game-theoretical value wins or losses, because, in a terminal-game state, the value is already proven. When at least one of the child nodes is proven to lose, or proven losing, the node is proven winning; when at all children nodes are proven winning, the node is proven losing [9]. In such a case, the proven node gets the reward either as $-\infty$ or $+\infty$, a lose and win, respectively. Then, a node with a proven node tries to deduce the evaluation.

3 The experimental results in games

3.1 Go

Since Brügmann [2] described the possibility of usage Monte-Carlo Search for the game Go with his program Gobble, there were several experiments conducted, the first of which was conducted on a 9x9 board. Despite the fact that that the implementation of eyes was slightly different than in Gobble, and adding Progressive Pruning improved the random games generation speed, it was not sufficient to outperform GNU Go, which was based on an alpha-beta search and domain-dependent knowledge [10]; in parallel, the tests showed that adding domain-knowledge such as $3x3$ patterns increased the performance of Monte-Carlo Search when compared with a random Monte-Carlo Search, likewise a move-generator using Progressive Pruning [11]. Moreover, comparing to a random Monte-Carlo Search, the domain-knowledge did search in the boards with more intersections (13x13 and 19x19) more effective.

The major breakthrough was in 2007, when Rémi Coulom, unlike his predecessors, first combined Tree Search with the Monte-Carlo method in his Crazy Stone program, which was also successfully used in other games [12]. The approach was calculating the value of all nodes (unlike Monte-Carlo Search, which passed only the evaluation of the end of nodes) by the formula below and pass them through root:

$$u_i = \exp(-2.4\frac{\mu_0 - \mu_i}{\sqrt{2(\sigma_0^2 + \sigma_i^2)}}) + \epsilon_i \tag{3}$$

where u_i is the urgency of move i, μ_i is the estimated value of move i ($\mu_0 > \mu_i$), σ_i^2 is a variance of move i, and ϵ_i is a constant which equals to

$$\epsilon_i = \frac{0.1 + 2^{-i} + a_i}{N} \tag{4}$$

and prevents the urgency of a move becoming equal to zero. In the formula, N equals to the total number of analyzed moves, and a_i equals to 1 in atari and 0 in otherwise. Another successful test was with the MoGo program, which, unlike Crazy Stone, was based on the UCT program [13]. In the first experiment of this program, the RAVE search combined with MCTS and UCT outperformed against GNUGo with a winning percentage above 52%; in the second approach, adding heuristic functions outplayed GNUGo with a winning percentage from 69% up to 97%. As a result of the researches, nowadays, all top Go Programs are based on the MCTS search.

3.2 Lines of Actions

For the game of Lines of Actions, MC-LOA program used MCTS with UCT, Progressive Bias and mixed strategy, containing several strategies such as:

- Evaluation Cut-Off to terminate the simulated game with high percentage.
- Corrective Strategy to minimize the risk of dubious moves;
- Greedy Strategy to reduce the number of analyzed moves.

It should be noticed that, instead of dividing to the number of games, Progressive Bias used $f(l_{j_1})$ function dividing number of losses in the node j_1:

$f(l_{j_1}) = \frac{k_{bias} \; * \; P_{mc}}{l_{j_1}+1}$

where k_{bias} is a constant, P_{mc} l_{j_1} is the transition probability of a move category mc, and 1 is a dominator to avoid the division to 0.

Experiments with MIA III, MIA 4.5, and the default version of MC-LOA described threshold value of evaluation cut-off equal to 700 as the best. An MC-LOA with mixed strategy performed better against other strategies. While MC-LOA with mixed strategy solved more endgame positions than non-variable-depth alpha-beta search, it could not outperform classic alpha-beta search of MIA 4.5. In direct matches against single-thread MIA 4.5, single-threaded MC-LOA scored almost with 47 ±3.1%. However, increasing the number of threads even up to 2 in MC-LOA rose the winning percentage of single-thread MIA 4.5 to 56%, because the efficiency of MC-LOA's search increased during parallelization. Moreover, recycling significant part of tree and using $\sqrt{l_{j_1} + 1}$ instead of l_{j_1} rose the winning percentage of single-threaded MC-LOA with mixed strategy to 52%, making Progressive Bias more relevant[14].

3.3 Hex

The MoHex program, which used MCTS with some enhancements, such as a parallel solver based on depth-first proof number (DFPN) search, experienced high success, winning the Computer Olympiad in 2009. In the experimental tournaments, it scored more than 70% against program Six and equal results against another program, Wolve. Both opponents used the alpha-beta approach to find the best move [15]. Another experimental research that used the Rapid Action Value Estimation (RAVE) algorithm described an inverse proportionality between a winning percentage and the value of UCT constant when the RAVE algorithm is used [16].

3.4 Havannah

There were conducted several kinds of research independent of each other:

1) The results of research by Stankiewicz, Winands, and Uiterwijk [17] and their simulations against default MCTS for Havannah showed that:

    - The Last-Good-Reply (LGR) generally improves the performance of MCTS. The best results occurred in the strategy called Last-Good-Reply with forgetting-1 (LGRF-1) when LGR did not test the move that leads to a loss in the playouts. (up to 62% winning percentage).
    - N-gram Average Sampling Technique also improves the performance of the MCTS search (up to 61% winning percentage). Moreover, the combination of LGR and N-gram together improved the performance better than separately, but the growth was slight when compared (up to 66% winning percentage).
    - Enhancing the selection step in MCTS by initialing the win counts and visit counts of new nodes with the values based on the results of corresponding moves, and biasing the search to certain types of moves, such as identifying joint and neighbour moves or local connections or edge and corner connections raises the performance of MCTS search (up to 69% winning percentage); Combination of these methods increased the winning up to 73% winning percentage.
    - However, the most dramatic increase was when both the enhancement combined by the combination of LGRF-2 (similar to LGRF-1, but with forgetting two previous moves) and N-gram strategy with decay value γequal to 0 and ϵ-greedy policy equal to 0.1 (up to 77.6% winning percentage). However, the

results also described that using only identifying joint and neighbor moves as enhancement might be enough in the combination of methods.

2) Joris Duguépéroux, Ahmad Mazyad, Fabien Teytaud, Julien Dehos [18] and their simulations against default MCTS for Havannah showed that:

   - The performance of MCTS and RAVE might be increased by choosing the proper value of constant K in formulas 1 and 3 described above. Additionally, RAVE search described better results against classical MCTS.

   - Progressive History, which is a combination of both Progressive Bias and History Heuristic, performed better than RAVE and scored up to 73% against standard MCTS algorithm.
   - Extended RAVE, which is a combination of both RAVE and Progressive History, was hardly slower than RAVE and outperformed RAVE with a winning rate.
   - RAVE with Havannah-Mate strategy, which finds immediate wins in 4 half-moves during playouts, required much more time than MCTS with random playouts and didn't give any significant advantages in long games. Implementing such fact and using Havannah-Mate for the first-third of playouts gave almost 60% against RAVE with random playouts.
   - The most significant improvement was when Progressive History used with Havannah-Mate; such a strategy scored almost 70% against classical MCTS. Another attractive approach by the same authors [18] used Progressive Widening, which was described in the article "Playout Pruning with Rave" (PPR). Experiments described the superiority of PPR against other playout improvements, such as RAVE, PoolRAVE, MAST, NAST, and LGRF1; the superiority increased proportionality to the size of the board.

3.5 Amazons

Using only UCT search increased Vanilla's win rate up to 81.6% against standard MCTS algorithm, while without UCT, the best result of the combination of average score and Win-Loss ratio was only 63.5%. However, the results dramatically increase with an accessibility evaluation for Amazons, which is a probability of accessing a square [19].

3.6 Chess

Despite the fact that the combination of algorithms, such as decisive moves, progressive bias, Heavy Playouts with ϵ-greedy policy equal to 0.4 and Static Exchange Evaluator, which examines all captures in a square, and endgame tablebases increased the strength of MCTS program for chess up to 864 Elo points compared to standard MCTS algorithm; the significant part of improvement was made by Heavy Playouts and Decisive moves. However, it was proven that even with such improvements, MCTS is not suitable to apply in chess and compete against the standard alpha-beta algorithm [20]. The main reason for this is the speed of the search algorithm, which could not identify search traps. Another problem described by standard MCTS algorithm was that the results of simulations were based on positional values, rather than the material of sides. Furthermore, standard MCTS had a difficulty to end a decisive advantage with a mate in simulations and ends them draw; this is solved by decreasing value simulations ended drawn. Despite such results, after the success of the AlphaGo program [21], which used MCTS for reinforcement Learning, MCTS is used by some robust programs as an enhancement of alpha-beta algorithm, which is a subject of future improvements. According to Mark Lefler, who is the one of authors Komodo 13.2.5, the idea is using short alpha-beta search in the child nodes of tree search and updating the values of nodes and root [22].

3.7 Shogi

MCTS search engine, using various methods such as assigning an Elo rating to moves, human games and an open book to define the weight of moves, progressive widening, killer moves, and history heuristics, was able to win only 4% percent of games against TOHMI program, which didn't use MCTS methods and was based on a search and evaluation function [23]. There were two possible reasons for this failure:

1) The evaluation of minor tactical losses, which put the program in a long-term disadvantage;
2) When MCTS was in situations with a disadvantage, it made move decisions that counted as inferior in TOHMI. The quiescence search reducing win rate was used to deal with this problem: when the quiescence search was used, the win rate rose to 32%.

Despite the results against TOHMI in games, and the results of tactical tests, which wasn't as high TOHMI, the MCTS search increased the quality of opening moves, and showed an endgame evaluation which most Shogi programs couldn't be enabled to reach [23].

4 Conclusion

MCTS, with its enhancements, is a robust search algorithm, which might sufficiently compete against an alpha-beta search in board games. However, while MCTS outperforms alpha-beta in strategical games as Hex, Lines of Actions, and Amazons due to the number of simulations, implementing MCTS in the tactical games with the highest number of search traps, such as Shogi and Go does not outperform the classic alpha-beta search. In addition to different enhancements of MCTS, in some games, such as Havannah and Go, the heuristic knowledge is required.

References

[1] Bruce D. Abramson. *The Expected-outcome Model of Two-player Games.* PhD thesis, New York, NY, USA, 1987. AAI8827528.

[2] Bernd Brügmann. Monte Carlo Go. 1993.

[3] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo Planning. In *ECML-06. Number 4212 in LNCS*, pages 282–293. Springer, 2006.

[4] Guillaume Chaslot, Mark H. M. Winands, H. Jaap van den Herik, Jos W. H. M. Uiterwijk, and Bruno Bouzy. Progressive Strategies for Monte-Carlo Tree Search. 2008.

[5] Rémi Coulom. Computing "Elo Ratings" of Move Patterns in the Game of Go. *ICGA Journal*, 30:198–208, 2007.

[6] Fabien Teytaud and Olivier Teytaud. On the huge benefit of decisive moves in Monte-Carlo Tree Search algorithms. *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, pages 359–364, 2010.

[7] Yizao Wang and Sylvain Gelly. Modifications of UCT and sequence-like simulations for Monte-Carlo Go. *2007 IEEE Symposium on Computational Intelligence and Games*, pages 175–182, 2007.

[8] Mark H. M. Winands and Yngvi Björnsson. Evaluation Function Based Monte-Carlo LOA. In *ACG*, 2009.

[9] Mark H. M. Winands, Yngvi Björnsson, and Jahn-Takeshi Saito. Monte-Carlo Tree Search Solver. pages 25–36, 2008.

[10] Bruno Bouzy and Bernard Helmstetter. Monte-Carlo Go Developments. In *ACG*, 2003.

[11] Bruno Bouzy. Associating domain-dependent knowledge and Monte Carlo approaches within a go program. *Inf. Sci.*, 175:247–257, 2005.

[12] Rémi Coulom. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In *Computers and Games*, 2006.

[13] Sylvain Gelly and David Silver. Monte-Carlo Tree Search and Rapid Action Value Estimation in Computer Go. *Artif. Intell.*, 175:1856–1875, 2011.

[14] Mark H. M. Winands, Yngvi Björnsson, and Jahn-Takeshi Saito. Monte Carlo Tree Search in Lines of Action. *IEEE Transactions on Computational Intelligence and AI in Games*, 2:239–250, 2010.

[15] Broderick Arneson, Ryan B. Hayward, and Philip Henderson. Monte Carlo Tree Search in Hex. *IEEE Transactions on Computational Intelligence and AI in Games*, 2:251–258, 2010.

[16] Tristan Cazenave and Abdallah Saffidine. Monte-Carlo Hex. 2010.

[17] Jan A. Stankiewicz, Mark H. M. Winands, and Jos W. H. M. Uiterwijk. Monte-Carlo Tree Search Enhancements for Havannah. In *ACG*, 2011.

[18] Joris Duguépéroux, Ahmad Mazyad, Fabien Teytaud, and Julien Dehos. Pruning Playouts in Monte-Carlo Tree Search for the Game of Havannah. In *Computers and Games*, 2016.

[19] Julien Kloetzer, Hiroyuki Iida, and Bruno Bouzy. The Monte-Carlo Approach in Amazons. 2007.

[20] Oleg Arenz. Monte Carlo Chess. Master's thesis, Knowledge Engineering Group, TU Darmstadt, 2012. Bachelor's Thesis.

[21] 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. *ArXiv*, abs/1712.01815, 2017.

[22] Komodo MCTS (Monte Carlo Tree Search) is the new star of TCEC, http://www.chessdom.com/komodo-mcts-monte-carlo-tree-search-is-the-new-star-of-tcec/.

[23] Yoshikuni Sato, Daisuke Takahashi, and Reijer Grimbergen. A Shogi Program Based on Monte-Carlo Tree Search. *ICGA Journal*, 33:80–92, 2010.