M

Monte-Carlo Tree Search

Mark H. M. Winands
Department of Data Science and Knowledge Engineering, Maastricht University, Maastricht, The Netherlands

Synonyms

MCTS; Monte-Carlo Tree Search; UCT

Definition

Monte-Carlo Tree Search (MCTS) (Coulom 2007; Kocsis et al. 2006) is a best-first search method that does not require a positional evaluation function. It is based on a randomized exploration of the search space. Using the results of previous explorations, the algorithm gradually builds up a game tree in memory and successively becomes better at accurately estimating the values of the most promising moves. MCTS consists of four strategic steps, repeated as long as there is time left (Chaslot et al. 2008b). The steps, outlined in Fig. 1, are as follows:

1. In the *selection step*, the tree is traversed from the root node downward until a state is chosen, which has not been stored in the tree.
2. Next, in the *play-out step*, moves are chosen in self-play until the end of the game is reached.
3. Subsequently, in the *expansion step*, one or more states encountered along its play-out are added to the tree.
4. Finally, in the *backpropagation step*, the game result r is propagated back along the previously traversed path up to the root node, where node statistics are updated accordingly.

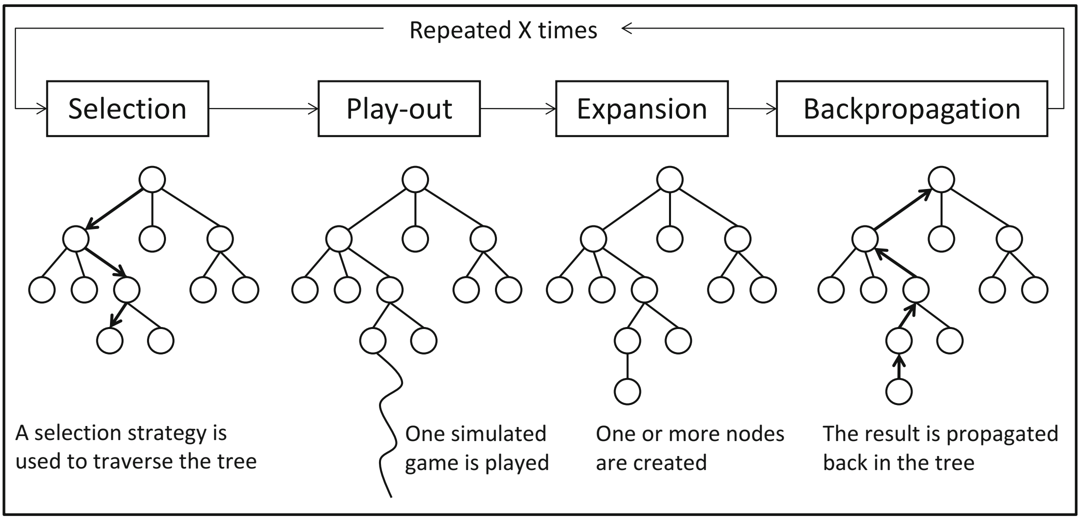
Structure of MCTS

MCTS usually starts with a tree containing only the root node. The tree is gradually grown by executing the selection, play-out, expansion, and backpropagation steps. Such an iteration is called a full simulation. After a certain number of simulations, a move is chosen to be played in the actual game. This final move selection is based on the highest score or alternatively the number of times being sampled. The detailed structure of MCTS is discussed by explaining the four steps below.

Selection

Selection chooses a child to be searched based on previous information. It controls the balance between exploitation and exploration. On the one hand, the task consists of selecting the move that leads to the best results so far (exploitation). On the other hand, the less promising moves still have to be tried, due to the uncertainty of the simulations (exploration).

© Springer International Publishing Switzerland 2015
N. Lee (ed.), *Encyclopedia of Computer Graphics and Games*,
DOI 10.1007/978-3-319-08234-9_12-1



Monte-Carlo Tree Search, Fig. 1 Outline of Monte-Carlo Tree Search (adapted from Chaslot et al. 2008b; Winands et al. 2010)

Several *selection strategies* (Browne et al. 2012) have been suggested for MCTS such as BAST, EXP3, and UCB1-Tuned, but the most popular one is based on the UCB1 algorithm (Auer et al. 2002), called UCT (**U**pper **C**onfidence Bounds applied to **T**rees) (Kocsis et al. 2006). UCT works as follows. Let I be the set of nodes immediately reachable from the current node p. The selection strategy selects the child b of node p that satisfies Formula 1:

$$b \in \operatorname{argmax}_{i \in I}\left(v_i + C \times \sqrt{\frac{\ln n_p}{n_i}}\right) \qquad (1)$$

where v_i is the value of the node i, n_i is the visit count of i, and n_p is the visit count of p. C is a parameter constant, which can be tuned experimentally (e.g., $C = 0.4$). The value of v_i should lie in the range [0, 1]. In case a child has not been stored in the tree or has not been visited yet, a default value is assumed. For example, the maximum value that a node could obtain by sampling (i.e., $v_{\max} = 1$) is taken.

Play-Out

When in the selection step a state is chosen, which has not been stored in the tree, the play-out starts. Moves are selected in self-play until the end of the game is reached. This task might consist of playing plain random moves or – better – semi-random moves chosen according to a *simulation strategy*. Smart simulation strategies have the potential to improve the level of play significantly. The main idea is to play interesting moves based on heuristics. In the literature this play-out step is sometimes called the roll-out or simulation.

Expansion

Expansion is the procedure that decides whether nodes are added to the tree. Standard the following *expansion strategy* is sufficient in most cases: one node is added per simulation (Coulom 2007). The added leaf node L corresponds to the first state encountered during the traversal that was not already stored. This allows to save memory and reduces only slightly the level of play.

Backpropagation

Backpropagation is the procedure that propagates the result r of a simulated game t back from the leaf node L, through the previously traversed nodes, all the way up to the root. If a game is won, the result of a player j is scored as $r_{t,j} = 1$, in the case of a loss as $r_{t,j} = 0$, and a draw as $r_{t,j} = 0.5$. To deal with multiplayer games, the result is backpropagated as a tuple of size N, where N is the number of players. For instance, if Player 1 and Player 3 both reach a winning condition in a 3-player game, then the result r is returned as the tuple $\left(\frac{1}{2}, 0, \frac{1}{2}\right)$. Propagating the values back in the tree is performed similar to max^n (Sturtevant 2008).

To compute the *value* v_i of a node i, a *backpropagation strategy* is applied. Usually, it is calculated by taking the average of the results of all simulated games made through this node (Coulom 2007), i.e., $v_i \leftarrow R_{i,j}/n_i$, where j is the player to move in its parent node p and $R_{i,j} \leftarrow \sum_t r_{t,j}$ the cumulative score of all the simulations.

MCTS Enhancements

Over the past years, several enhancements have been developed to improve the performance of MCTS (Browne et al. 2012). First, there are many ways to improve the selection step of MCTS. The major challenge is how to choose a promising node when the number of simulations is still low. Domain-independent techniques that only use information gathered during the simulations are Transposition Tables, Rapid Action Value Estimation (RAVE), and Progressive History (Childs et al. 2008; Gelly et al. 2012; Nijssen and Winands 2011). Techniques that rely on hand-coded domain knowledge are, for instance, Move Groups, Prior Knowledge, Progressive Bias, and Progressive Widening/Unpruning (Chaslot et al. 2008b; Childs et al. 2008; Gelly et al. 2012). The used heuristic knowledge may consist of move patterns and even static board evaluators. When a couple of these enhancements are successfully incorporated, the C parameter of UCT becomes usually very small or even zero.

Next, the play-outs require a simulation strategy in order to be accurate. Moves are chosen based on only computationally light knowledge (Gelly et al. 2012) (e.g., patterns, capture potential, and proximity to the last move). Adding computationally intensive heavy heuristic knowledge in the play-outs (such as a 1- or 2-ply search using a full board evaluator) has been beneficial in a few games such as Chinese Checkers and Lines of Action. When domain knowledge is not readily available, there exist various domain-independent techniques to enhance the quality of the play-outs, including the Move-Average Sampling Technique (MAST), Last-Good-Reply Policy, and N-Grams (Tak et al. 2012). The principle of these techniques is that moves good in one situation are likely to be good in other situations as well.

The basic version of MCTS *converges* to the game-theoretic value, but is unable to prove it. The MCTS-Solver technique (Winands et al. 2010) is able to prove the game-theoretic value of a state with a binary outcome (i.e., win or loss). It labels terminal states in the search tree as a win or loss and backpropagates the game-theoretic result in a max^n way (Nijssen and Winands 2011). For games with multiple outcomes (e.g., win, loss, or draw), the technique has been extended to Score Bounded Monte-Carlo Tree Search (Cazenave and Saffidine 2011).

Finally, to utilize the full potential of a multi-core machine, parallelization has to be applied in an MCTS program. There exist three different parallelization techniques for MCTS: (1) *root parallelization*, (2) *leaf parallelization*, and (3) *tree parallelization* (Chaslot et al. 2008a). In root parallelization, each thread has its own MCTS tree. When the allotted search time is up, the results of the different trees are combined. In leaf parallelization, one tree is traversed using a single thread. Subsequently, starting from the leaf node, play-outs are executed in parallel for each available thread. Once all threads have finished, the results are backpropagated. When using tree parallelization, one tree is shared, in which all threads operate independently. For shared memory systems, tree parallelization is the natural approach that takes full advantage of the available bandwidth to communicate simulation results (Enzenberger and Müller 2010).

Historical Background

Classic search algorithms such as A*, $\alpha\beta$ search, or Expectimax require an evaluator that assigns heuristic values to the leaf nodes in the tree. The 15-puzzle and the board games backgammon, chess, and checkers are instances where this approach has led to world-class performance. However, for some domains constructing a strong static heuristic evaluation function has been a rather difficult or an even infeasible task.

Replacing such an evaluation function with Monte-Carlo sampling was proposed in the early 1990s. Abramson (1990) experimented with these so-called Monte-Carlo evaluations in the games of tic-tac-toe, Othello, and chess. In 1993 Bernd Brügmann was the first to use Monte-Carlo evaluations in his 9 × 9 Go program Gobble. The following years, the technique was incorporated in stochastic games such as backgammon (Tesauro et al. 1997) and imperfect-information games such as bridge (Ginsberg 1999), poker (Billings et al. 1999), and Scrabble (Sheppard 2002).

In the early 2000s, the Monte-Carlo approach received new interest in the Computer Go domain (Bouzy and Helmstetter 2004). Bruno Bouzy's Monte-Carlo Go engine Indigo had some limited success as the main challenge was to effectively combine Monte-Carlo evaluations with game-tree search. The breakthrough came when Coulom presented the MCTS approach at the 2006 Computers and Games Conference (Coulom 2007). He subsequently demonstrated its strength by winning the 9 × 9 Go tournament at the 12th ICGA Computer Olympiad with his MCTS engine Crazy Stone. Simultaneously Kocsis and Szepesvári (Kocsis et al. 2006) introduced the MCTS variant UCT. Its selection strategy became the standard for many MCTS engines (Browne et al. 2012). Techniques such as RAVE, Prior Knowledge, Progressive Bias, and Progressive Widening (Chaslot et al. 2008b; Gelly et al. 2012) were needed to make MCTS effective in many challenging domains such as 19 × 19 Go. Parallelization (Enzenberger et al. 2010; Gelly et al. 2012) has enabled MCTS to compete with human Go Grandmasters. As of 2014, an MCTS engine can beat a 9-dan professional player with only a four-stone handicap, whereas a decade ago 20 stones could be given.

Applications

In the past few years, MCTS has substantially advanced the state of the art in several abstract games (Browne et al. 2012), in particular Go (Gelly et al. 2012), but other two-player deterministic perfect-information games include Amazons (Lorentz 2008), Hex (Arneson et al. 2010), and Lines of Action (Winands et al. 2010). MCTS has even increased the level in multiplayer games such as Chinese checkers (Sturtevant 2008) and games with stochasticity and/or imperfect information such as Kriegspiel (Ciancarini and Favini 2010), Lord of the Rings: The Confrontation (Cowling et al. 2012), and Scotland Yard (Nijssen and Winands 2012). In the General Game Playing competition, where an agent has to play many different abstract games without any human intervention, MCTS has become the dominant approach as well (Björnsson and Finnsson 2009).

Besides application to abstract games, MCTS has made inroads in the video game domain. It has been applied in the arcade game Ms. Pac-Man for controlling either the Ghosts or the Pac-Man (Nguyen and Thawonmas 2013; Pepels et al. 2014). The technique has been used for resource allocation and coordination in the turn-based strategy game Total War: Rome II and for tactical assault planning in the real-time strategy game Wargus (Balla et al. 2009). The MCTS framework has also shown promise in the General Video Game AI Competition (Perez et al. 2014), where the goal is to build an agent that is capable of playing a wide range of (simple) video games.

MCTS has also been applied in puzzle games such as SameGame (Schadd et al. 2012) where it is hard to design an admissible evaluation function for A* or IDA*. As these games are close to scheduling and optimization problems, MCTS has been introduced in real-life applications. They are, for instance, high energy physics (Ruijl et al. 2014), patient admission scheduling (Zhu et al. 2014), and interplanetary trajectory planning (Hennes et al. 2015).

Future Directions

MCTS does not require a positional evaluation function, overcoming partially the knowledge acquisition bottleneck. It is therefore a promising method when an agent has to play a wide range of games as is fostered in the General (Video) Game Playing competitions. However, for MCTS to work effectively, search-control knowledge is required to guide the simulations. Domain-independent techniques are able to boost the decision quality of an MCTS engine, but for achieving expert level hand-coded domain knowledge is incorporated to grasp high-level context. Instead of being hand-coded by the programmer, a future research direction is to automatically discover, extract, represent, and tune this control knowledge during online search.

MCTS has been quite successful in abstract games; however, the number of successful applications in modern video games with high fidelity is rather limited. There are three challenges for applying MCTS in these games. (1) In these video games, the action space is large if not infinite, and the state space is often continuous. For MCTS to work effectively, the game world has to be abstracted automatically in such a way that (i) the number of possible moves is limited and (ii) the number of moves required to finish the game is reduced as well. (2) These games have a high degree of uncertainty, not only due to non-determinism (the outcome of a move cannot be predicted) or imperfect information (certain information is hidden for a player) but also because of incomplete information (the behavior of the physics engine may be unknown). For non-determinism and imperfect information, MCTS enhancements have been investigated to a limited number of abstract games (Cowling et al. 2012), but even less for video games. Dealing with incomplete information in the MCTS framework is a largely unexplored terrain. (3) Due to the real-time property the amount of deliberation time is limited. MCTS has to generate a sufficient number of simulations in a short time as otherwise the decision quality is quite low (Björnsson and Finnsson 2009).

References and Further Reading

Abramson, B.: Expected-outcome: A general model of static evaluation. IEEE Trans. Pattern Anal. Mach. Intell. **12**(2), 182–193 (1990)

Arneson, B., Hayward, R.B., Henderson, P.: Monte Carlo Tree Search in Hex. IEEE Trans. Comput. Intell. AI Games **2**(4), 251–258 (2010)

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

Balla, R.K., Fern A.: UCT for tactical assault planning in real-time strategy games. In: Boutilier, C. (ed.) Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09), pp. 40–45. AAAI Press, Menlo Park, CA, USA (2009)

Billings, D., Peña, L., Schaeffer, J., Szafron, D.: Using probabilistic knowledge and simulation to play poker. In: Hendler, J., Subramanian, D. (eds) Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence, pp. 697–703. AAAI Press/The MIT Press, Menlo Park, CA, USA (1999)

Björnsson, Y., Finnsson, H.: CadiaPlayer: A simulation-based General Game Player. IEEE Trans. Comput. Intell. AI Games **1**(l), 4–15 (2009)

Bouzy, B., Helmstetter, B.: Monte-Carlo Go developments. In: van den Herik, H.J., Iida, H., Heinz, E.A. (eds.) Advances in Computer Games 10: Many Games, Many Challenges. IFIP Advances in Information and Communication Technology, vol. 135, pp. 159–174. Kluwer, Boston (2004)

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

Cazenave, T., Saffidine, A.: Score bounded Monte-Carlo Tree Search. In: van den Herik, H.J., Iida, H., Plaat, A. (eds.) Computers and Games (CG 2010). Lecture Notes in Computer Science, vol. 6515, pp. 93–104. Springer, Berlin (2011)

Chaslot, G.M.J.-B., Winands, M.H.M., van den Herik, H. J.: Parallel Monte-Carlo Tree Search. In: van den Herik, H.J., Xu, X., Ma, Z., Winands, M.H.M. (eds.) Computers and Games (CG 2008). Lecture Notes in Computer Science, vol. 5131, pp. 60–71. Springer, Berlin (2008a)

Chaslot, G.M.J.-B., Winands, M.H.M., van den Herik, H. J., Uiterwijk, J.W.H.M., Bouzy, B.: Progressive strategies for Monte-Carlo Tree Search. New Math. Nat. Comput. **4**(3), 343–357 (2008b)

Childs, B.E., Brodeur, J.H., Kocsis, L.: Transpositions and move groups in Monte Carlo Tree Search. In: Hingston, P., Barone, L. (eds.) Proceedings of the 2008 IEEE Symposium on Computational Intelligence and Games, pp. 389–395. IEEE, Piscataway, NJ, USA (2008)

Ciancarini, P., Favini, G.P.: Monte Carlo Tree Search in Kriegspiel. AI J. **174**(11), 670–684 (2010)

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.) Computers and Games (CG 2006). Lecture Notes in Computer Science, vol. 4630, pp. 72–83. Springer, Berlin (2007)

Cowling, P.I., Powley, E.J., Whitehouse, D.: Information set Monte Carlo Tree Search. IEEE Trans. Comput. Intell. AI Games **4**(2), 120–143 (2012)

Enzenberger, M., Müller, M.: A lock-free multithreaded Monte-Carlo Tree Search algorithm. In: van den Herik, H.J., Spronck, P. (eds.) Advances in Computer Games (ACG 2009). Lecture Notes in Computer Science (LNCS), vol. 6048, pp. 14–20. Springer, Berlin (2010)

Enzenberger, M., Müller, M., Arneson, B., Segal, R.: Fuego – an open-source framework for board games and Go engine based on Monte Carlo Tree Search. IEEE Trans. Comput. Intell AI Games **2**(4), 259–270 (2010)

Gelly, S., Kocsis, L., Schoenauer, M., Sebag, M., Silver, D., Szepesvári, C., Teytaud, O.: The grand challenge of computer Go: Monte Carlo Tree Search and extensions. Commun. ACM **55**(3), 106–113 (2012)

Ginsberg, M.L.: GIB: Steps toward an expert-level bridge-playing program. In: Dean, T. (ed.) Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99), vol. 1, pp. 584–589. Morgan Kaufmann, San Francisco, CA, USA (1999)

Hennes, D., Izzo, D.: Interplanetary trajectory planning with Monte Carlo Tree Search. In: Yang, Q., Wooldridge, M. (eds.) Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015), pp. 769–775. AAAI Press, Menlo Park, CA, USA (2015)

Kocsis, L., Szepesvári, C.: Bandit based Monte-Carlo Planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) Machine Learning: ECML 2006. Lecture Notes in Artificial Intelligence, vol. 4212, pp. 282–293. Springer, Berlin (2006)

Lorentz, R.J.: Amazons discover Monte-Carlo. In: van den Herik, H.J., Xu, X., Ma, Z., Winands, M.H.M. (eds.) Computers and Games (CG 2008). Lecture Notes in Computer Science, vol. 5131, pp. 13–24. Springer, Berlin (2008)

Nguyen, K.Q., Thawonmas, R.: Monte Carlo Tree Search for collaboration control of Ghosts in Ms. Pac-Man. IEEE Trans. Comput. Intell. AI Games **5**(1), 57–68 (2013)

Nijssen, J.A.M., Winands, M.H.M.: Enhancements for multi-player Monte-Carlo Tree Search. In: van den Herik, H.J., Iida, H., Plaat, A. (eds.) Computers and Games (CG 2010). Lecture Notes in Computer Science, vol. 6151, pp. 238–249. Springer, Berlin (2011)

Nijssen, J.A.M., Winands, M.H.M.: Monte Carlo Tree Search for the hide-and-seek game Scotland Yard. Trans. Comput. Intell. AI Games **4**(4), 282–294 (2012)

Pepels, T., Winands, M.H.M., Lanctot, M.: Real-time Monte Carlo Tree Search in Ms Pac-Man. IEEE Trans. Comput. Intell. AI Games **6**(3), 245–257 (2014)

Perez, D., Samothrakis, S., Lucas, S.M.: Knowledge-based fast evolutionary MCTS for general video game playing. In: Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG 2014), pp. 68–75 (2014)

Ruijl, B., Vermaseren, J., Plaat, A. van den Herik, H.J.: Combining simulated annealing and Monte Carlo Tree Search for expression simplification. In: ICAART 2014, pp. 724–731 (2014)

Schadd, M.P.D., Winands, M.H.M., Tak, M.J.W., Uiterwijk, J.W.H.M.: Single-player Monte-Carlo Tree Search for SameGame. Knowl.-Based Syst. **34**, 3–11 (2012)

Sheppard, B.: World-championship-caliber Scrabble. Artif. Intell. **134**(1–2), 241–275 (2002)

Sturtevant, N.R.: An analysis of UCT in multi-player games. ICGA J. **31**(4), 195–208 (2008)

Tak, M.J.W., Winands, M.H.M., Björnsson, Y.: N-Grams and the last-good-reply policy applied in general game playing. IEEE Trans. Comput. Intell. AI Games **4**(2), 73–83 (2012)

Tesauro, G., Galperin, G.R.: On-line policy improvement using Monte-Carlo search. In: Mozer, M.C., Jordan, M.I., Petsche, T. (eds.) Advances in Neural Information Processing Systems, vol. 9, pp. 1068–1074. MIT Press, Cambridge, MA, USA (1997)

Winands, M.H.M., Björnsson, Y., Saito, J.-T.: Monte Carlo Tree Search in Lines of Action. IEEE Trans. Comput. Intell. AI Games **2**(4), 239–250 (2010)

Zhu, G., Lizotte, D., Hoey, J.: Scalable approximate policies for Markov decision process models of hospital elective admissions. Artif. Intell. Med. **61**(1), 21–34 (2014)