Monte Carlo tree search

In computer science, **Monte Carlo tree search** (**MCTS**) is a heuristic search algorithm for some kinds of decision processes, most notably those employed in game play. The leading examples of Monte Carlo tree search are recent computer Go programs,[1] followed by chess and shogi,[2] as well as real-time video games (such as Total War: Rome II's implementation in the high level campaign AI[3]) and games with incomplete information such as bridge[4] and poker.[5]

Monte Carlo tree search	
Class	Search algorithm

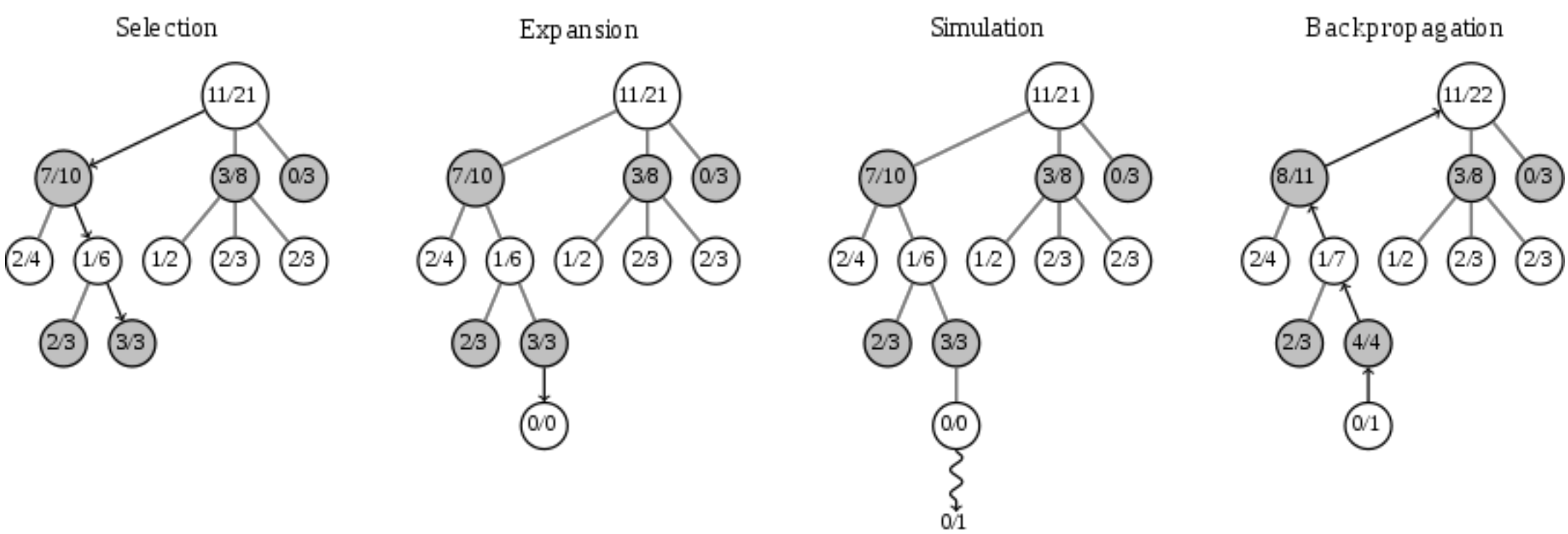
Contents

Principle of operation

The focus of Monte Carlo tree search is on the analysis of the most promising moves, expanding the search tree based on random sampling of the search space. The application of Monte Carlo tree search in games is based on many *playouts*. In each playout, the game is played out to the very end by selecting moves at random. The final game result of each playout is then used to weight the nodes in the game tree so that better nodes are more likely to be chosen in future playouts.

The most basic way to use playouts is to apply the same number of playouts after each legal move of the current player, then choose the move which led to the most victories.[6] The efficiency of this method—called *Pure Monte Carlo Game Search*—often increases with time as more playouts are assigned to the moves that have frequently resulted in the current player's victory according to previous playouts. Each round of Monte Carlo tree search consists of four steps:[7]

- *Selection*: start from root R and select successive child nodes until a leaf node L is reached. The root is the current game state and a leaf is any node from which no simulation (playout) has yet been initiated. The section below says more about a way of biasing choice of child nodes that lets the game tree expand towards the most promising moves, which is the essence of Monte Carlo tree search.
- *Expansion*: unless L ends the game decisively (e.g. win/loss/draw) for either player, create one (or more) child nodes and choose node C from one of them. Child nodes are any valid moves from the game position defined by L.
- *Simulation*: complete one random playout from node C. This step is sometimes also called playout or rollout. A playout may be as simple as choosing uniform random moves until the game is decided (for example in chess, the game is won, lost, or drawn).
- *Backpropagation*: use the result of the playout to update information in the nodes on the path from C to R.



Steps of Monte Carlo tree search

This graph shows the steps involved in one decision, with each node showing the ratio of wins to total playouts from that point in the game tree for the player that node represents.[8] In the Selection diagram, black is about to move. The root node shows there are 11 wins out of 21 playouts for white from this position so far. It complements the total of 10/21 black wins shown along the three black nodes under it, each of which represents a possible black move.

If white loses the simulation, all nodes along the selection incremented their simulation count (the denominator), but among them only the black nodes were credited with wins (the numerator). If instead white wins, all nodes along the selection would still increment their simulation count, but among them only the white nodes would be credited with wins. In games where draws are possible, a draw causes the numerator for both black and white to be incremented by 0.5 and the denominator by 1. This ensures that during selection, each player's choices expand towards the most promising moves for that player, which mirrors the goal of each player to maximize the value of their move.

Rounds of search are repeated as long as the time allotted to a move remains. Then the move with the most simulations made (i.e. the highest denominator) is chosen as the final answer.

Pure Monte Carlo game search

This basic procedure can be applied to any game whose positions necessarily have a finite number of moves and finite length. For each position, all feasible moves are determined: *k* random games are played out to the very end, and the scores are recorded. The move leading to the best score is chosen. Ties are broken by fair coin flips. Pure Monte Carlo Game Search results in strong play in several games with random elements, as in the game *EinStein würfelt nicht!*. It converges to optimal play (as *k* tends to infinity) in board filling games with random turn order, for instance in Hex with random turn order.[9]

Exploration and exploitation

The main difficulty in selecting child nodes is maintaining some balance between the *exploitation* of deep variants after moves with high average win rate and the *exploration* of moves with few simulations. The first formula for balancing exploitation and exploration in games, called UCT (*Upper Confidence Bound* 1 *applied to trees*), was introduced by Levente Kocsis and Csaba Szepesvári.[10] UCT is based on the UCB1 formula derived by Auer, Cesa-Bianchi, and Fischer[11] and the provably convergent AMS (Adaptive Multi-stage Sampling) algorithm first applied to multi-stage decision making models (specifically, Markov Decision Processes) by Chang, Fu, Hu, and Marcus.[12] Kocsis and Szepesvári recommend to choose in each node of the game tree the move for which the expression $\frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}}$ has the highest value. In this formula:

- w_i stands for the number of wins for the node considered after the i-th move
- n_i stands for the number of simulations for the node considered after the i-th move
- N_i stands for the total number of simulations after the i-th move
- c is the exploration parameter—theoretically equal to $\sqrt{2}$; in practice usually chosen empirically

The first component of the formula above corresponds to exploitation; it is high for moves with high average win ratio. The second component corresponds to exploration; it is high for moves with few simulations.

Most contemporary implementations of Monte Carlo tree search are based on some variant of UCT that traces its roots back to the AMS simulation optimization algorithm for estimating the value function in finite-horizon Markov Decision Processes (MDPs) introduced by Chang et al.[12] (2005) in Operations Research. (AMS was the first work to explore the idea of UCB-based exploration and exploitation in constructing sampled/simulated (Monte Carlo) trees and was the main seed for UCT.[13])

Advantages and disadvantages

Although it has been proven that the evaluation of moves in Monte Carlo tree search converges to minimax,[14] the basic version of Monte Carlo tree search converges very slowly. However Monte Carlo tree search does offer significant advantages over alpha–beta pruning and similar algorithms that minimize the search space.

In particular, Monte Carlo tree search does not need an explicit evaluation function. Simply implementing the game's mechanics is sufficient to explore the search space (i.e. the generating of allowed moves in a given position and the game-end conditions). As such, Monte Carlo tree search can be employed in games without a developed theory or in general game playing.

The game tree in Monte Carlo tree search grows asymmetrically as the method concentrates on the more promising subtrees. Thus it achieves better results than classical algorithms in games with a high branching factor.

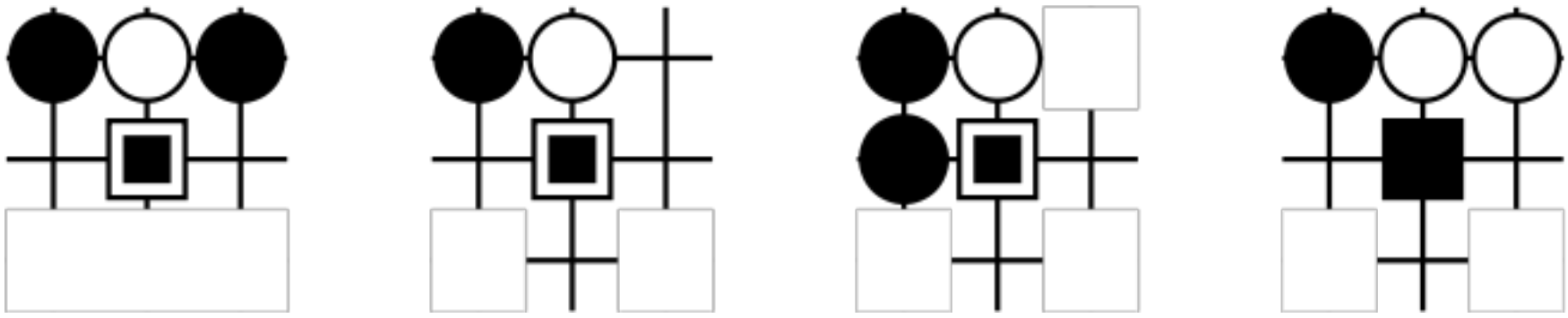
Moreover, Monte Carlo tree search can be interrupted at any time yielding the most promising move already found.

A disadvantage is that, faced in a game with an expert player, there may be a single branch which leads to a loss. Because this is not easily found at random, the search may not "see" it and will not take it into account. It is believed that this may have been part of the reason for AlphaGo's loss in its fourth game against Lee Sedol. In essence, the search attempts to prune sequences which are less relevant. In some cases, a play can lead to a very specific line of play which is significant, but which is overlooked when the tree is pruned, and this outcome is therefore "off the search radar".[15]

Improvements

Various modifications of the basic Monte Carlo tree search method have been proposed to shorten the search time. Some employ domain-specific expert knowledge, others do not.

Monte Carlo tree search can use either *light* or *heavy* playouts. Light playouts consist of random moves while heavy playouts apply various heuristics to influence the choice of moves.[16] These heuristics may employ the results of previous playouts (e.g. the Last Good Reply heuristic[17]) or expert knowledge of a given game. For instance, in many Go-playing programs certain stone patterns in a portion of the board influence the probability of moving into that area.[18] Paradoxically, playing suboptimally in simulations sometimes makes a Monte Carlo tree search program play stronger overall.[19]

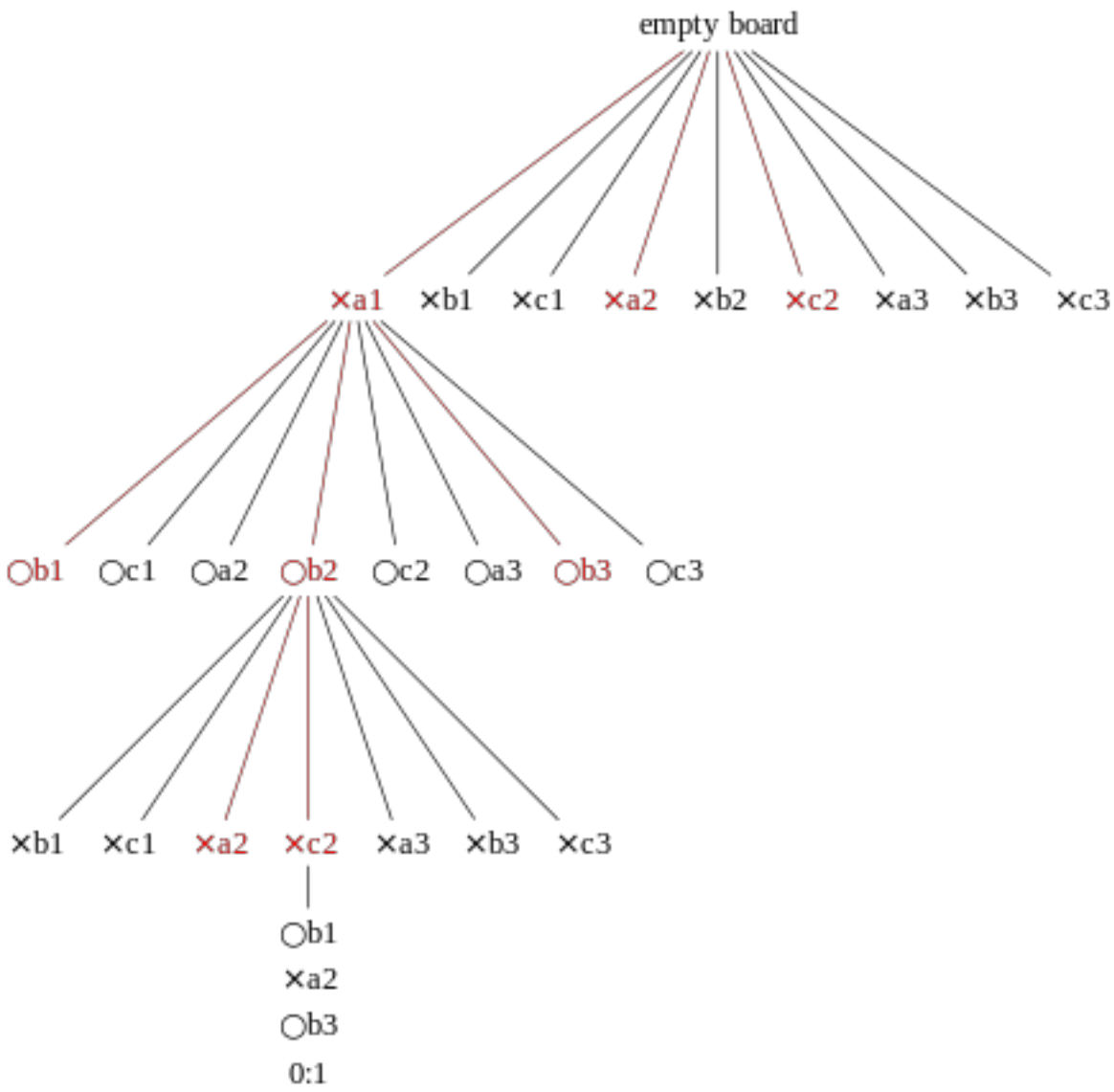


Patterns of *hane* (surrounding opponent stones) used in playouts by the MoGo program. It is advantageous for both black and white to put a stone on the middle square, except the rightmost pattern where it favors black only.[18]

Domain-specific knowledge may be employed when building the game tree to help the exploitation of some variants. One such method assigns nonzero *priors* to the number of won and played simulations when creating each child node, leading to artificially raised or lowered average win rates that cause the node to be chosen more or less frequently, respectively, in the selection step.[20] A related method, called *progressive bias*, consists in adding to the UCB1 formula a $\frac{b_i}{n_i}$ element, where b_i is a heuristic score of the i-th move.[7]

The basic Monte Carlo tree search collects enough information to find the most promising moves only after many rounds; until then its moves are essentially random. This exploratory phase may be reduced significantly in a certain class of games using RAVE (*Rapid Action Value Estimation*).[20] In these games, permutations of a sequence of moves lead to the same position. Typically, they are board games in which a move involves placement of a piece or a stone on the board. In such games the value of each move is often only slightly influenced by other moves.

In RAVE, for a given game tree node N, its child nodes C_i store not only the statistics of wins in playouts started in node N but also the statistics of wins in all playouts started in node N and below it, if they contain move i (also when the move was played in the tree, between node N and a playout). This way the contents of tree nodes are influenced not only by moves played immediately in a given position but also by the same moves played later.



RAVE on the example of tic-tac-toe. In red nodes, the RAVE statistics will be updated after the b1-a2-b3 simulation.

When using RAVE, the selection step selects the node, for which the modified UCB1 formula $(1-\beta(n_i,\tilde{n}_i))\frac{w_i}{n_i}+\beta(n_i,\tilde{n}_i)\frac{\tilde{w}_i}{\tilde{n}_i}+c\sqrt{\frac{\ln t}{n_i}}$ has the highest value. In this formula, $\tilde{w}_i$ and $\tilde{n}_i$ stand for the number of won playouts containing move i and the number of all playouts containing move i, and the $\beta(n_i,\tilde{n}_i)$ function should be close to one and to zero for relatively small and relatively big n_i and $\tilde{n}_i$, respectively. One of many formulas for $\beta(n_i,\tilde{n}_i)$, proposed by D. Silver,[21] says that in balanced positions one can take $\beta(n_i,\tilde{n}_i)=\frac{\tilde{n}_i}{n_i+\tilde{n}_i+4b^2n_i\tilde{n}_i}$, where b is an empirically chosen constant.

Heuristics used in Monte Carlo tree search often require many parameters. There are automated methods to tune the parameters to maximize the win rate.[22]

Monte Carlo tree search can be concurrently executed by many threads or processes. There are several fundamentally different methods of its parallel execution:[23]

- *Leaf parallelization*, i.e. parallel execution of many playouts from one leaf of the game tree.
- *Root parallelization*, i.e. building independent game trees in parallel and making the move basing on the root-level branches of all these trees.
- *Tree parallelization*, i.e. parallel building of the same game tree, protecting data from simultaneous writes either with one, global mutex, with more mutexes, or with non-blocking synchronization.[24]

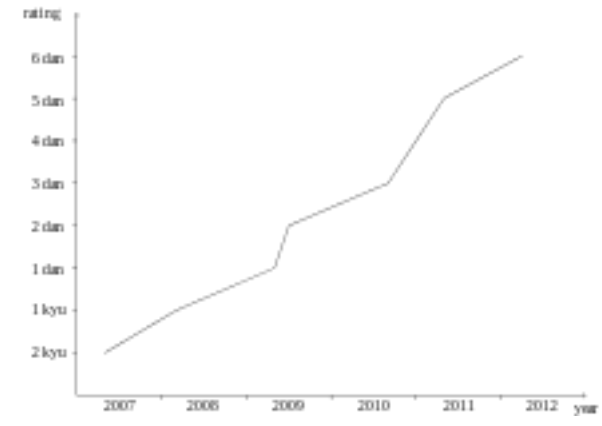
History

The Monte Carlo method, which uses randomness for deterministic problems difficult or impossible to solve using other approaches, dates back to the 1940s. Bruce Abramson explored the MCTS idea in his 1987 PhD thesis and said it "is shown to be precise, accurate, easily estimable, efficiently calculable, and domain-independent."[25] He experimented in-depth with Tic-tac-toe and then with machine-generated evaluation functions for Othello and Chess.

Such methods were then explored and successfully applied to heuristic search in the field of automated theorem proving by W. Ertel, J. Schumann and C. Suttner in 1989,[26][27][28] thus improving the exponential search times of uninformed search algorithms such as e.g. breadth-first search, depth-first search or iterative deepening.

In 1992, B. Brügmann employed it for the first time in a Go-playing program.[6] Chang et al.[12] proposed the idea of "recursive rolling out and backtracking" with "adaptive" sampling choices in their Adaptive Multi-stage Sampling (AMS) algorithm for the model of Markov decision processes. (AMS was the first work to explore the idea of UCB-based exploration and exploitation in constructing sampled/simulated (Monte Carlo) trees and was the main seed for UCT.[13])

In 2006, inspired by these predecessors,[30] Rémi Coulom described the application of the Monte Carlo method to game-tree search and coined the name Monte Carlo tree search,[31] L. Kocsis and Cs. Szepesvári developed the UCT algorithm,[10] and S. Gelly et al. implemented UCT in their program MoGo.[18] In 2008, MoGo achieved dan (master) level in 9×9 Go,[32] and the Fuego program began to win with strong amateur players in 9×9 Go.[33]



The rating of best Go-playing programs on the KGS server since 2007. Since 2006, all the best programs use Monte Carlo tree search.[29]

In January 2012, the Zen program won 3:1 in a Go match on a 19×19 board with an amateur 2 dan player.[34] Google Deepmind developed the program AlphaGo, which in October 2015 became the first Computer Go program to beat a professional human Go player without handicaps on a full-sized 19x19 board.[1][35][36] In March 2016, AlphaGo was awarded an honorary 9-dan (master) level in 19×19 Go for defeating Lee Sedol in a five-game match with a final score of four games to one.[37] AlphaGo represents a significant improvement over previous Go programs as well as a milestone in machine learning as it uses Monte Carlo tree search with artificial neural networks (a deep learning method) for policy (move selection) and value, giving it efficiency far surpassing previous programs.[38]

Monte Carlo tree search has also been used in programs that play other board games (for example Hex,[39] Havannah,[40] Game of the Amazons,[41] and Arimaa[42]), real-time video games (for instance Ms. Pac-Man[43][44] and Fable Legends), and nondeterministic games (such as skat,[45] poker,[5] Magic: The Gathering,[46] or Settlers of Catan[47]).

See also

- AlphaGo, a Go program using Monte Carlo tree search, reinforcement learning and deep learning.
- AlphaGo Zero, an updated Go program using Monte Carlo tree search, reinforcement learning and deep learning.
- AlphaZero, a generalized version of AlphaGo Zero using Monte Carlo tree search, reinforcement learning and deep learning.
- Leela Chess Zero, a public domain application of AlphaZero methods to chess, which is currently among the leading chess playing programs.

References

1. Silver, David; Huang, Aja; Maddison, Chris J.; Guez, Arthur; Sifre, Laurent; Driessche, George van den; Schrittwieser, Julian; Antonoglou, Ioannis; Panneershelvam, Veda; Lanctot, Marc; Dieleman, Sander; Grewe, Dominik; Nham, John; Kalchbrenner, Nal; Sutskever, Ilya; Lillicrap, Timothy; Leach, Madeleine; Kavukcuoglu, Koray; Graepel, Thore; Hassabis, Demis (28 January 2016). "Mastering the game of Go with deep neural networks and tree search" (https://www.nature.com/nature/journal/v529/n7587/full/nature16961.html). *Nature*. **529** (7587): 484–489. Bibcode:2016Natur.529..484S (http://adsabs.harvard.edu/abs/2016Natur.529..484S). doi:10.1038/nature16961 (https://doi.org/10.1038%2Fnature16961). ISSN 0028-0836 (https://www.worldcat.org/issn/0028-0836). PMID 26819042 (https://www.ncbi.nlm.nih.gov/pubmed/26819042). Retrieved 10 December 2017.
2. Silver, David (2017). "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm". arXiv:1712.01815v1 (https://arxiv.org/abs/1712.01815v1).
3. "Monte-Carlo Tree Search in TOTAL WAR: ROME II's Campaign AI" (http://aigamedev.com/open/coverage/mcts-rome-ii/). *AI Game Dev*. Retrieved 25 February 2017.
4. Stuart J. Russell, Peter Norvig (2009). *Artificial Intelligence: A Modern Approach* (3rd ed.). Prentice Hall.
5. Jonathan Rubin; Ian Watson (April 2011). "Computer poker: A review" (https://web.archive.org/web/20120813081731/https://www.cs.auckland.ac.nz/~jrub001/files/CPReviewPreprintAIJ.pdf) (PDF). *Artificial Intelligence*. **175** (5–6): 958–987. doi:10.1016/j.artint.2010.12.005 (https://doi.org/10.1016%2Fj.artint.2010.12.005).
6. Brügmann, Bernd (1993). *Monte Carlo Go* (http://www.ideanest.com/vegos/MonteCarloGo.pdf) (PDF). Technical report, Department of Physics, Syracuse University.
7. G.M.J.B. Chaslot; M.H.M. Winands; J.W.H.M. Uiterwijk; H.J. van den Herik; B. Bouzy (2008). "Progressive Strategies for Monte-Carlo Tree Search" (https://dke.maastrichtuniversity.nl/m.winands/documents/pMCTS.pdf) (PDF). *New Mathematics and Natural Computation*. **4** (3): 343–359. doi:10.1142/s1793005708001094 (https://doi.org/10.1142%2Fs1793005708001094).
8. Bradberry, Jeff (2015-09-07). "Introduction to Monte Carlo Tree Search" (http://jeffbradberry.com/posts/2015/09/intro-to-monte-carlo-tree-search/).
9. Peres, Yuval; Schramm, Oded; Sheffield, Scott; Wilson, David B. (2006). "Random-Turn Hex and other selection games". arXiv:math/0508580 (https://arxiv.org/abs/math/0508580).

10. Kocsis, Levente; Szepesvári, Csaba (2006). "Bandit based Monte-Carlo Planning" (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.102.1296). In Fürnkranz, Johannes; Scheffer, Tobias; Spiliopoulou, Myra. *Machine Learning: ECML 2006, 17th European Conference on Machine Learning, Berlin, Germany, September 18–22, 2006, Proceedings*. Lecture Notes in Computer Science. **4212**. Springer. pp. 282–293. doi:10.1007/11871842_29 (https://doi.org/10.1007%2F11871842_29). ISBN 3-540-45375-X.
11. Auer, Peter; Cesa-Bianchi, Nicolò; Fischer, Paul (2002). "Finite-time Analysis of the Multiarmed Bandit Problem" (http://moodle.technion.ac.il/pluginfile.php/192340/mod_resource/content/0/UCB.pdf) (PDF). *Machine Learning*. **47**: 235–256. doi:10.1023/a:1013689704352 (https://doi.org/10.1023%2Fa%3A1013689704352).
12. Chang, Hyeong Soo; Fu, Michael C.; Hu, Jiaqiao; Marcus, Steven I. (2005). "An Adaptive Sampling Algorithm for Solving Markov Decision Processes" (http://scholar.rhsmith.umd.edu/sites/default/files/mfu/files/cfhm05.pdf?m=1449834091) (PDF). *Operations Research*. **53**: 126–139. doi:10.1287/opre.1040.0145 (https://doi.org/10.1287%2Fopre.1040.0145).
13. Hyeong Soo Chang; Michael Fu; Jiaqiao Hu; Steven I. Marcus (2016). "Google DeepMind's Alphago: O.R.'s unheralded role in the path-breaking achievement" (https://www.informs.org/ORMS-Today/Public-Articles/October-Volume-43-Number-5). *ORMS Today*. INFORMS. **45** (5): 24–29.
14. Bouzy, Bruno. "Old-fashioned Computer Go vs Monte-Carlo Go". *IEEE Symposium on Computational Intelligence and Games, April 1–5, 2007, Hilton Hawaiian Village, Honolulu, Hawaii* (http://ewh.ieee.org/cmte/cis/mtsc/ieeecis/tutorial2007/Bruno_Bouzy_2007.pdf) (PDF).
15. "Lee Sedol defeats AlphaGo in masterful comeback - Game 4" (https://gogameguru.com/lee-sedol-defeats-alphago-masterful-comeback-game-4/). Go Game Guru.
16. Swiechowski, M.; Mandziuk, J., "Self-Adaptation of Playing Strategies in General Game Playing" (2010), *IEEE Transactions on Computational Intelligence and AI in Games*, doi: 10.1109/TCIAIG.2013.2275163, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6571225&isnumber=4804729
17. Drake, Peter (December 2009). "The Last-Good-Reply Policy for Monte-Carlo Go". *ICGA Journal*. **32** (4): 221–227.
18. Sylvain Gelly; Yizao Wang; Rémi Munos; Olivier Teytaud (November 2006). *Modification of UCT with Patterns in Monte-Carlo Go* (http://hal.inria.fr/docs/00/11/72/66/PDF/MoGoReport.pdf) (PDF). Technical report, INRIA.
19. Seth Pellegrino; Peter Drake (2010). "Investigating the Effects of Playout Strength in Monte-Carlo Go". *Proceedings of the 2010 International Conference on Artificial Intelligence, ICAI 2010, July 12–15, 2010, Las Vegas Nevada, USA*. Hamid R. Arabnia, David de la Fuente, Elena B. Kozerenko, José Angel Olivas, Rui Chang, Peter M. LaMonica, Raymond A. Liuzzi, Ashu M. G. Solo (eds.). CSREA Press. pp. 1015–1018. ISBN 1-60132-148-1.
20. Sylvain Gelly; David Silver (2007). "Combining Online and Offline Knowledge in UCT". *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20–24, 2007* (http://www.machinelearning.org/proceedings/icml2007/papers/387.pdf) (PDF). Zoubin Ghahramani (ed.). ACM. pp. 273–280. ISBN 978-1-59593-793-3.
21. David Silver (2009). *Reinforcement Learning and Simulation-Based Search in Computer Go* (http://papersdb.cs.ualberta.ca/~papersdb/uploaded_files/1029/paper_thesis.pdf) (PDF). PhD thesis, University of Alberta.
22. Rémi Coulom. "CLOP: Confident Local Optimization for Noisy Black-Box Parameter Tuning". *ACG 2011: Advances in Computer Games 13 Conference, Tilburg, the Netherlands, November 20–22* (http://remi.coulom.free.fr/CLOP/).
23. Guillaume M.J-B. Chaslot, Mark H.M. Winands, Jaap van den Herik (2008). "Parallel Monte-Carlo Tree Search". *Computers and Games, 6th International Conference, CG 2008, Beijing, China, September 29 – October 1, 2008. Proceedings* (https://dke.maastrichtuniversity.nl/m.winands/documents/multithreadedMCTS2.pdf) (PDF). H. Jaap van den Herik, Xinhe Xu, Zongmin Ma, Mark H. M. Winands (eds.). Springer. pp. 60–71. ISBN 978-3-540-87607-6.
24. Markus Enzenberger; Martin Müller (2010). "A Lock-free Multithreaded Monte-Carlo Tree Search Algorithm" (http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=02CC0F88A12A3CCE44F0CD139ADA7AF5?doi=10.1.1.161.1984&rep=rep1&type=pdf). In Jaap Van Den Herik; Pieter Spronck. *Advances in Computer Games: 12th International Conference, ACG 2009, Pamplona, Spain, May 11–13, 2009, Revised Papers*. Springer. pp. 14–20. ISBN 978-3-642-12992-6.
25. Abramson, Bruce (1987). *The Expected-Outcome Model of Two-Player Games* (http://academiccommons.columbia.edu/download/fedora_content/download/ac:142327/CONTENT/CUCS-315-87.pdf) (PDF). Technical report, Department of Computer Science, Columbia University. Retrieved 23 December 2013.
26. Wolfgang Ertel; Johann Schumann; Christian Suttner (1989). "Learning Heuristics for a Theorem Prover using Back Propagation.". In J. Retti; K. Leidlmair. *5. Österreichische Artificial-Intelligence-Tagung. Informatik-Fachberichte 208,pp. 87-95* (http://www.hs-weingarten.de/~ertel/veroeff_bib.html#ESS89). Springer.
27. Christian Suttner; Wolfgang Ertel (1990). "Automatic Acquisition of Search Guiding Heuristics.". *CADE90, 10th Int. Conf. on Automated Deduction.pp. 470-484. LNAI 449* (http://www.hs-weingarten.de/~ertel/veroeff_bib.html#ES90:CADE). Springer.
28. Christian Suttner; Wolfgang Ertel (1991). "Using Back-Propagation Networks for Guiding the Search of a Theorem Prover" (http://www.hs-weingarten.de/~ertel/veroeff_bib.html#ES90:IJNN). *Journal of Neural Networks Research & Applications*. **2** (1): 3–16.
29. "Sensei's Library: KGSBotRatings" (http://senseis.xmp.net/?KGSBotRatings). Retrieved 2012-05-03.
30. Rémi Coulom (2008). "The Monte-Carlo Revolution in Go". *Japanese-French Frontiers of Science Symposium* (http://remi.coulom.free.fr/JFFoS/JFFoS.pdf) (PDF).
31. Rémi Coulom (2007). "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search". *Computers and Games, 5th International Conference, CG 2006, Turin, Italy, May 29–31, 2006. Revised Papers* (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.81.6817). H. Jaap van den Herik, Paolo Ciancarini, H. H. L. M. Donkers (eds.). Springer. pp. 72–83. ISBN 978-3-540-75537-1.

32. Chang-Shing Lee; Mei-Hui Wang; Guillaume Chaslot; Jean-Baptiste Hoock; Arpad Rimmel; Olivier Teytaud; Shang-Rong Tsai; Shun-Chin Hsu; Tzung-Pei Hong (2009). "The Computational Intelligence of MoGo Revealed in Taiwan's Computer Go Tournaments" (http://hal.inria.fr/docs/00/36/97/86/PDF/TCIAIG-2008-0010_Accepted_.pdf) (PDF). *IEEE Transactions on Computational Intelligence and AI in Games*. **1** (1): 73–89. doi:10.1109/tciaig.2009.2018703 (https://doi.org/10.1109%2Ftciaig.2009.2018703).
33. Markus Enzenberger; Martin Mūller (2008). *Fuego – An Open-Source Framework for Board Games and Go Engine Based on Monte Carlo Tree Search* (http://pug.raph.free.fr/files/Fuego.pdf) (PDF). Technical report, University of Alberta.
34. "The Shodan Go Bet" (http://dcook.org/gobet/). Retrieved 2012-05-02.
35. "Research Blog: AlphaGo: Mastering the ancient game of Go with Machine Learning" (http://googleresearch.blogspot.com/2016/01/alphago-mastering-ancient-game-of-go.html). *Google Research Blog*. 27 January 2016.
36. "Google achieves AI 'breakthrough' by beating Go champion" (https://www.bbc.com/news/technology-35420579). *BBC News*. 27 January 2016.
37. "Match 1 - Google DeepMind Challenge Match: Lee Sedol vs AlphaGo" (https://www.youtube.com/watch?v=vFr3K2DORc8&t=1h57m). *Youtube*. 9 March 2016.
38. "Google AlphaGo AI clean sweeps European Go champion" (http://www.zdnet.com/article/google-alphago-ai-clean-sweeps-european-go-champion/). *ZDNet*. 28 January 2016.
39. Broderick Arneson; Ryan Hayward; Philip Henderson (June 2009). "MoHex Wins Hex Tournament" (http://webdocs.cs.ualberta.ca/~hayward/papers/rptPamplona.pdf) (PDF). *ICGA Journal*. **32** (2): 114–116.
40. Timo Ewalds (2011). *Playing and Solving Havannah* (http://havannah.ewalds.ca/static/thesis.pdf) (PDF). Master's thesis, University of Alberta.
41. Richard J. Lorentz (2008). "Amazons Discover Monte-Carlo". *Computers and Games, 6th International Conference, CG 2008, Beijing, China, September 29 – October 1, 2008. Proceedings*. H. Jaap van den Herik, Xinhe Xu, Zongmin Ma, Mark H. M. Winands (eds.). Springer. pp. 13–24. ISBN 978-3-540-87607-6.
42. Tomáš Kozelek (2009). *Methods of MCTS and the game Arimaa* (http://arimaa.com/arimaa/papers/TomasKozelekThesis/mt.pdf) (PDF). Master's thesis, Charles University in Prague.
43. Xiaocong Gan; Yun Bao; Zhangang Han (December 2011). "Real-Time Search Method in Nondeterministic Game – Ms. Pac-Man". *ICGA Journal*. **34** (4): 209–222.
44. Tom Pepels; Mark H. M. Winands; Marc Lanctot (September 2014). "Real-Time Monte Carlo Tree Search in Ms Pac-Man" (http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6731713). *IEEE Transactions on Computational Intelligence and AI in Games*. **6** (3): 245–257. doi:10.1109/tciaig.2013.2291577 (https://doi.org/10.1109%2Ftciaig.2013.2291577).
45. Michael Buro; Jeffrey Richard Long; Timothy Furtak; Nathan R. Sturtevant (2009). "Improving State Evaluation, Inference, and Search in Trick-Based Card Games". *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11–17, 2009* (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.150.3077). Craig Boutilier (ed.). pp. 1407–1413.
46. C.D. Ward; P.I. Cowling (2009). "Monte Carlo Search Applied to Card Selection in Magic: The Gathering". *CIG'09 Proceedings of the 5th international conference on Computational Intelligence and Games* (https://web.archive.org/web/20160528074031/http://scim.brad.ac.uk/staff/pdf/picowlin/CIG2009.pdf) (PDF). IEEE Press. Archived from the original (http://scim.brad.ac.uk/staff/pdf/picowlin/CIG2009.pdf) (PDF) on 2016-05-28.
47. István Szita; Guillaume Chaslot; Pieter Spronck (2010). "Monte-Carlo Tree Search in Settlers of Catan" (http://ticc.uvt.nl/icga/acg12/proceedings/Contribution100.pdf) (PDF). In Jaap Van Den Herik; Pieter Spronck. *Advances in Computer Games, 12th International Conference, ACG 2009, Pamplona, Spain, May 11–13, 2009. Revised Papers*. Springer. pp. 21–32. ISBN 978-3-642-12992-6.

Bibliography

- Cameron Browne; Edward Powley; Daniel Whitehouse; Simon Lucas; Peter I. Cowling; Philipp Rohlfshagen; Stephen Tavener; Diego Perez; Spyridon Samothrakis; Simon Colton (March 2012). "A Survey of Monte Carlo Tree Search Methods". *IEEE Transactions on Computational Intelligence and AI in Games*. **4** (1). doi:10.1109/tciaig.2012.2186810 (https://doi.org/10.1109%2Ftciaig.2012.2186810).