The Mathematics Behind the Monte Carlo Tree Search Algorithm

Artificial Intelligence and Machine Learning are terms that have taken over the internet, conjuring images of digital brains and intelligent code. But how is it, exactly, that these algorithms are created? Is it truly possible for ones and zeros to think and reason? Or are they simply following rules which allow them to predict what their output should be?

This question was prevalent around the world in March of 2016, when DeepMind (the creator of AlphaGo) pitted the AI against the best human Go player, Lee Sedol [9]. AlphaGo's 4-1 win was a major milestone in machine learning, not simply for its victory over humans, but also in the way that it won.

Go is famous for being a game of not just strategy, but also intuition [9]. Lee Sedol is known for his 'creative fighting play': something people thought that AlphaGo would never be able to achieve. This view was challenged by move 37 - a move that was played during the second match against Lee Sedol [9]. This was a move that baffled all the Go experts observing the game. One commentator remarked that it was a "totally unthinkable move", and another said, "I think we're seeing an original move here" [9].

One of the features of AlphaGo was that the creators could ask it to provide information on the move it had selected [9]. When questioned about move 37, AlphaGo said that there was a 1 in 10,000 probability that a human would have chosen that move [9]. This means that the move was not common in its database but, if so, then why did AlphaGo choose it? This brings us back to the question of how its decisions are made, to which there are three main factors [9]:

1. The Monte Carlo Tree Search algorithm
2. The policy network (scanning the board and coming up with possible areas to play in)
3. The value network (evaluating the board position)

In this essay, I shall focus on the Monte Carlo Tree Search algorithm.

The Monte Carlo Tree Search algorithm (MCTS) is a heuristic search algorithm [1]. Heuristic means an approach to problem solving that, although not necessarily optimal, is sufficient to get an estimate of the answer [2]. Heuristic algorithms are used when optimal solutions are impossible, or when they are impractical in terms of speed [2].

As stated in the DeepMind documentary [9], "the number of possible configurations of the board is more than the number of atoms in the universe". It is therefore impossible to get an optimal solution for Go (almost all possibilities would need to be checked), so a heuristic algorithm is required. Go is not the only game where the MCTS is used; others that employ it include Chess, Checkers, Scrabble, and Backgammon [1].

The goal of the MCTS is to analyse the most promising moves. The algorithm required to train it for this involves four main steps [1][4][5][6]:

1. Selection
2. Expansion
3. Simulation
4. Backpropagation

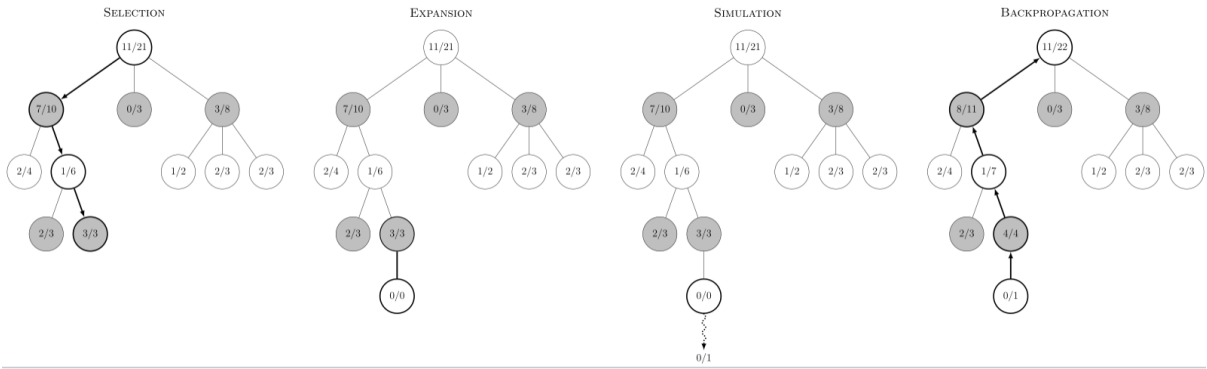


Diagram above comes from source 1: Monte Carlo Tree Search - Wikipedia

Selection:

In selection, a node must be chosen to analyse. The challenge that arises with this step is maintaining a balance between **exploitation** and **exploration** [1].

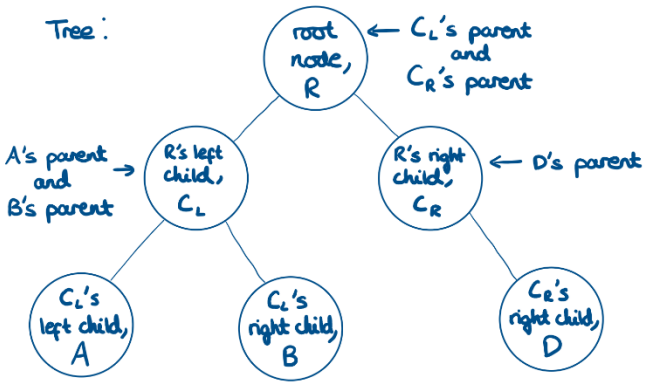
Exploitation is the analysis of simulations that occur after moves with a high average win rate.

Exploration is the analysis of moves which have had few simulations.

The node selected is the node for which the value of the following formula is maximised [1]:

Formula below comes from source 1: Monte Carlo Tree Search - Wikipedia

$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}}$$



- w_i is the number of wins for the node considered after the *i*-th move.
- n_i is the number of simulations for the node considered after the *i*-th move.
- N_i is the total number of simulations after the *i*-th move which has been run by the parent node of the one considered.
- c is the exploration constant. In theory, this is equal to $\sqrt{2}$. In practice, the algorithm is observed, and its value chosen manually.

w_i/n_i can also be written as $\bar{x}_i$ which stands for the mean value of that node after the *i*-th move [4]. The mean value is the number of wins divided by the number of games played, so in other words the average win rate. This component of the formula values exploitation as it is high for moves with a larger fraction of wins [4].

The second component (with the square root) values exploration. The fewer number of simulations that the node has had (n_i), the larger the second component [1].

This equation is a branch of solutions from the multi-armed bandit problem (named after one-armed bandits) [5][14][15]. The multi-armed bandit problem is a problem in which limited resources, such

as selecting moves, must be allocated between competing choices in order to maximise reward [15]. The property of each choice is only partially known at the beginning; however, more can be learnt about it from testing the choices [15].

Expansion:

Unless the selected node ends the game (win/lose/draw), child nodes are added to the tree (connected to the selected node). A random node, C, is chosen from them [1][11].

Simulation:

Complete a playout (run a simulation) from node C. There are two main ways to do this [1]:

- Random moves (also known as light playouts)
- Apply a variety of heuristics to influence what moves are chosen (heavy playouts). In this case, either the results of previous playouts or domain-specific expert knowledge would be required.

A method involving domain-specific knowledge is to assign values, prior to simulations, when creating a child node. This will influence the chance of it being picked as well as the average win rate of the node. For example, in Go, certain stone patterns in part of the board can affect the probability of moving into that area. [1]

Another method of selecting moves is to use Rapid Action Value Estimation (RAVE). This method is usually used in games which involve permutations of a sequence of moves resulting in the same position. [1]

The difference with RAVE is that each node, C_i, stores not only the statistics of wins in playouts originating from its parent, N, but also the statistics of wins in the playouts of all children of N which contain move *i*. This means that the impact of move *i* in other playouts is taken into account. [1][19]

The formula to maximise for selecting a node, when using RAVE, is changed to this [1]:

Formula below comes from source 1: Monte Carlo Tree Search - Wikipedia

$$(1-\beta(n_i,\tilde{n}_i))\frac{w_i}{n_i}+\beta(n_i,\tilde{n}_i)\frac{\tilde{w}_i}{\tilde{n}_i}+c\sqrt{\frac{\ln t}{n_i}}$$

- w_i is still the number of wins for the node considered after the *i*-th move.
- n_i is still the number of simulations for the node considered after the *i*-th move.
- $\tilde{w}_i$ is the number of wins for all playouts containing move *i*.
- $\tilde{n}_i$ is the number of simulations containing move *i*.
- $\beta(n_i, \tilde{n}_i)$ is a function that should be close to one for relatively small n_i and $\tilde{n}_i$ (which means that not many simulations have been seen) but close to zero for relatively big n_i and $\tilde{n}_i$ (which means that many simulations have been seen). What this does is help balance the exploration of the algorithm as the value of the formula will be lower for moves that have had lots of simulations ($\beta(n_i, \tilde{n}_i)$ is closer to zero). [1][19]
- The last component of the formula is the same as before, the N_i replaced with a *t*.

Backpropagation:

The result of the simulation (win/draw/lose) is used to update all the nodes leading from the child, C, back to the root node. For games which have black and white pieces, each node would represent either a black move or a white move.

If white, for example, loses a simulation, then all black nodes along the path can have w_i (number of wins) incremented by 1, all white nodes have w_i decreased by one, and all nodes have n_i (number of simulations played out) incremented by 1. If there is a draw, then for all nodes along the path (black and white) the value of w_i can remain the same (+0) while n_i increases by 1. [5][18]

Minimax:

It has been proven that the MCTS converges to **minimax** [1]. Minimax is when you have minimised the loss for the worst-case scenario [3]. The formula for minimax is [3]:

Formula below comes from source 3: MiniMax - Wikipedia

$$\overline{v_i} = \min_{a_{-i}} \max_{a_i} v_i(a_i, a_{-i})$$

- *i* is the index of the player of interest.
- a_i is the action taken by player *i.*
- a_{-i} is the action taken by all the other players.
- v_i is the value function of player *i.*

What this means is that for each possible move of the player, we figure out which one will minimise the maximum score of the other players. In a two-player game like Go, this is the same as maximising your own score.

Selecting final move:

The move played by the program after all its calculations can be chosen on the basis of four factors [18]:

1. Max child – the child with the highest value (won to played ratio)
2. Robust child – the child with the highest visit count
3. Robust-max-child – the child with both the highest visit count and the highest value. This is the optimal situation, so the simulations are likely to continue in the MCTS until this child appears.
4. Secure child – the child which maximises v + A/$\sqrt{n}$. Here, A is a constant (that is set manually), v is the value of the node, and n is the node visit count.

The best move found will be returned after a 'computational budget' has been reached [13]. For example, after a certain period of time has elapsed.

Conclusion:

Although we have learned how the Monte Carlo Tree Search algorithm works, the truth of how AlphaGo selects a move is a lot more complicated due to its use of neural networks. It is because of the complicated neural networks that, even now, the reason AlphaGo chose move 37 is still unknown. Was it a stroke of genius, or was it an error in the software? Analysing the MCTS can dampen the magical vision of AlphaGo being an intelligent entity able to think for itself. But let's not

forget: the human brain is simply a network of neurons able to pass electrical signals to one another - and is it not a wonderous thing?

Our brains were evolved over millions of years, so it's hardly a surprise that creating artificial intelligence is going to be a difficult task requiring incredible persistence and dedication. The algorithm I've talked about here is one of many, and I've only scratched the surface. People are working on using machine learning for robotics, for cybersecurity, for language processing models (we've all heard of ChatGPT), for healthcare, and for countless others.

Mathematics is an integral part of computer science, and it's no mystery why when learning about what algorithms and formulae lie behind the MCTS. It is my hope that minds equipped with a passion for both subjects can be inspired from great achievements like AlphaGo to push for a better, brighter future of technology.

Sources:

[1] Wikipedia Contributors, "Monte Carlo tree search," *Wikipedia*, Mar. 15, 2023. https://en.wikipedia.org/wiki/Monte_Carlo_tree_search (accessed Mar. 27, 2023).

[2] Wikipedia Contributors, "Heuristic," *Wikipedia*, Feb. 07, 2023. https://en.wikipedia.org/wiki/Heuristic (accessed Mar. 27, 2023).

[3] Wikipedia Contributors, "Minimax," *Wikipedia*, Jan. 25, 2023. https://en.wikipedia.org/wiki/Minimax (accessed Mar. 27, 2023).

[4] "ML Monte Carlo Tree Search MCTS," *GeeksforGeeks*, Jan. 14, 2019. https://www.geeksforgeeks.org/ml-monte-carlo-tree-search-mcts/ (accessed Mar. 27, 2023).

[5] B. Wang, "Monte Carlo Tree Search: An Introduction - Towards Data Science," *Medium*, Jan. 10, 2021. https://towardsdatascience.com/monte-carlo-tree-search-an-introduction-503d8c04e168 (accessed Mar. 27, 2023).

[6] "Monte-Carlo Tree Search (MCTS) — Introduction to Reinforcement Learning," *Github.io*, 2016. https://gibberblot.github.io/rl-notes/single-agent/mcts.html (accessed Mar. 27, 2023).

[7] http://localhost:4000/about.html, "Move 37: Artificial Intelligence, Randomness, and Creativity," *John Menick*, Oct. 17, 2016. https://johnmenick.com/writing/move-37-alpha-go-deep-mind.html (accessed Mar. 27, 2023).

[8] Wikipedia Contributors, "AlphaGo," *Wikipedia*, Feb. 13, 2023. https://en.wikipedia.org/wiki/AlphaGo (accessed Mar. 27, 2023).

[9] DeepMind, "AlphaGo - The Movie | Full award-winning documentary," *YouTube*. Mar. 13, 2020. Accessed: Mar. 27, 2023. [YouTube Video]. Available: https://www.youtube.com/watch?v=WXuK6gekU1Y

[10] for, “Computer Science learning for school students,” *Teach-ict.com*, 2023. https://teach-ict.com/2016/GCSE_Computing/AQA_8520/3_3_data_representation/huffman/miniweb/pg2.php (accessed Mar. 27, 2023).

[11] J. Bradberry, “Introduction to Monte Carlo Tree Search - Jeff Bradberry,” *Jeffbradberry.com*, Sep. 07, 2015. https://jeffbradberry.com/posts/2015/09/intro-to-monte-carlo-tree-search/ (accessed Mar. 27, 2023).

[12] A. Wong, “Solving the Multi-Armed Bandit Problem - Towards Data Science,” *Medium*, Sep. 25, 2017. https://towardsdatascience.com/solving-the-multi-armed-bandit-problem-b72de40db97c (accessed Mar. 27, 2023).

[13] M. How, “MCTS: How to choose the final action from the root,” *Artificial Intelligence Stack Exchange*, Dec. 03, 2019. https://ai.stackexchange.com/questions/16905/mcts-how-to-choose-the-final-action-from-the-root (accessed Mar. 27, 2023).

[14] L. Weng, “The Multi-Armed Bandit Problem and Its Solutions,” *Github.io*, Jan. 23, 2018. https://lilianweng.github.io/posts/2018-01-23-multi-armed-bandit/ (accessed Mar. 27, 2023).

[15] Wikipedia Contributors, “Multi-armed bandit,” *Wikipedia*, Feb. 28, 2023. https://en.wikipedia.org/wiki/Multi-armed_bandit (accessed Mar. 27, 2023).

[16] D. Silver *et al.*, “Mastering the game of Go without human knowledge,” *Nature*, 2017. https://www.semanticscholar.org/paper/Mastering-the-game-of-Go-without-human-knowledge-Silver-Schrittwieser/c27db32efa8137cbf654902f8f728f338e55cd1c#:~:text=An%20artificial-intelligence%20program%20called%20AlphaGo%20Zero%20has%20mastered,with%20Deep%20Reinforcement%20Learning%20Supported%20by%20Domain%20Knowledge (accessed Mar. 27, 2023).

[17] L. Kocsis and C. Szepesvári, “Bandit Based Monte-Carlo Planning,” *Lecture Notes in Computer Science*, pp. 282–293, 2006, doi: https://doi.org/10.1007/11871842_29.

[18] Guillaume Chaslot, Mark H.M. Winands, V. Den, and B. Bouzy, “Progressive Strategies for Monte-Carlo Tree Search,” *ResearchGate*, Nov. 2008. https://www.researchgate.net/publication/23751563_Progressive_Strategies_for_Monte-Carlo_Tree_Search (accessed Mar. 27, 2023).

[19] S. Gelly and D. Silver, “Monte-Carlo tree search and rapid action value estimation in computer Go,” *Artificial Intelligence*, vol. 175, no. 11, pp. 1856–1875, Jul. 2011, doi: https://doi.org/10.1016/j.artint.2011.03.007.