Monte Carlo Search Tree Algorithm (MCTS)

The Monte Carlo Search Tree algorithm gets its focus from an analysis of the most promising moves of the search tree, based on random sampling on the search tree. The MCTS is comfortable with making tradeoffs of how it uses its computation time in order to find the most optimal solutions.
MCTS embodies the strategy of using random trials to solve a problem and choosing the trials that give the best overall results in order to make the best possible decision. For this reason, perfect information games such as Chess, Go, Quoridor... where there is no hidden information from either player are the perfect targets for a Monte Carlo Tree Search algorithm, because everything is fully determined, a tree can be constructed that contains all possible outcomes, and a value assigned corresponding to a win or a loss for one of the players.

1 Design motivation

But why is MCTS a good algorithm for perfect information games? Because it avoids the principal bottlenecks or issues present on regular Minimax algorithms, being random by design, looking through different branching paths of their already big branching value considering their absurd number of possible moves which also helps them avoid bad decisions when they become so unlikely that the algorithm just gives up on checking them in order to save computation time. This also helps MCTS budget its computation time, since only the most likely outcomes are explored.
This causes another issue though, since if the move chosen is random every time, the probability of getting the best path forward becomes extremely unlikely. For that reason, statistics are kept from every move state into the next one, and the best overall result is chosen.

2 Upper Confidence Trees (UCT)

But how can we use the gathered statistics to decide which is the better path? By using a variation of the MCTS, the Upper Confidence Tree. When there are multiple branching paths, each with a different rate of success, a strategy that maximizes the overall net worth is desired. For that, all branches would need to be checked, but concentrating on the observable best path. One method to get the best path is using the Upper Confidence Bound.
Where is the exploitation component, the winning/losing rate, while is the number of times the parent node was visited in the tree and the number of times the current node(child) has been visited. This last part is the exploration component. The exploitation and exploration component are balanced to get a value that will lead the algorithm on the tree traversal. These two components are balanced by a factor that will be used to give more or less weight to each side of the formula.

3 Principles of Operation

The algorithm is divided into 4 stages. These 4 stages are executed in a loop as many times as the user desires. The loop is controlled either by number of iterations or by time. The more times it iterates through these 4 stages it will end up in better game moves. At the beginning of the algorithm, when the tree is yet to be created, since no simulations has started yet, unvisited nodes are chosen first. The algorithm will simulate each one of them and results will be back propagated to the root until this is fully expanded. This very beginning algorithm logic follows indeed the four principles of operation, but because the

tree is empty it is better to explain these 4 principles of operations in a more advanced situation, where the tree has some nodes already created.

- *Selection:*

In this first stage the algorithm will look for a leaf node that is yet to be visited. We will start from the root and successively select child nodes until a leaf node is reached. The root being the current game state and the leaf any node that has a potential child from where no simulation has been yet initiated. This tree traversal is done using the above explained formula the Upper Confidence Tree (UCT), which will balance between deep variants with high average win rate and other moves with few simulations.

- *Expansion:*

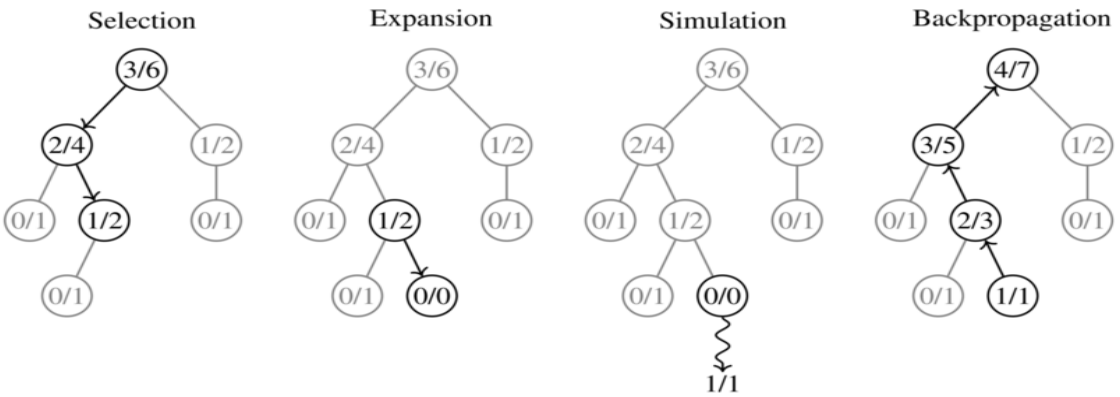
Once the leaf node has been selected, and after checking that it is not a terminal node, meaning that the game does not end in this node move, this' node children will be created. Children nodes will be any valid game moves from this node's game state. For example, in a tic-tac-toe game children nodes will be where our cross can be placed in the nine squares. It is important to remember that the children will be created once the node has been simulated at least once. In better words, node's children will not be created until the node has been selected in the first principle of operation at least once. In the case of creating new children one of them will be passed to the next stage. And in the case of not creating any children yet, the current node will be the one passed to the next stage.

- *Simulation*

This stage is where the game that the algorithm is implemented in takes more importance. In this stage the game will be simulated from the leaf node game state. The simulation will go on until a game ending condition is reached, either win, draw or lose. From that simulation data will be gathered and passed to the next stage. The simulation could go on by just doing random moves one after the other, or by having some game-design bias where some game logic is added into the simulation.

- *Back-Propagation*

This last stage is the easiest one. Here the data that the simulation has achieved will be propagated back from the leaf node expanded in the second principle of operation until the root node which is the current game state. If the simulation was good and a win game condition was achieved the nodes that lead to that condition will have an updated value in their win rate stat.



Monte Carlo Tree Search algorithm offers significant advantages over pruning and similar algorithms that minimize the search space. The tree that the algorithm expands grows asymmetrically as the method concentrates on promising subtrees. This is a clear advantage over other algorithms especially in games with high branching factor.

The biggest disadvantage that can be found in this algorithm is that certain position there may be moves that look superficially strong but end up leading to a loss. This is something that professional GO players do to beat the MCTS AI.

References:

https://en.wikipedia.org/wiki/Monte_Carlo_tree_search
https://www.youtube.com/watch?v=lhFXKNyA0QA
https://www.remi-coulom.fr/JFFoS/JFFoS.pdf
http://jeffbradberry.com/posts/2015/09/intro-to-monte-carlo-tree-search/