Monte Carlo Tree Search

MICHAEL WOLLOWSKI

COMPILED FROM THE FOLLOWING RESOURCES:

HTTPS://EN.WIKIPEDIA.ORG/WIKI/MONTE_CARLO_TREE_SEARCH

MONTE-CARLO TREE SEARCH: A NEW FRAMEWORK FOR GAME AI

Introduction

Monte Carlo tree search (**MCTS**) is a heuristic search algorithm

Used in in game play.

- MCTS was introduced in 2006 for computer Go
- Used in board games like chess and shogi
- Used in games with incomplete information such as bridge and poker
- Used in real-time video games such as Total War: Rome II's implementation of the high level campaign AI.

Principle of Operation

The focus of MCTS is on the analysis of the most promising moves.

It expands 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* also called *roll-outs*.

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.

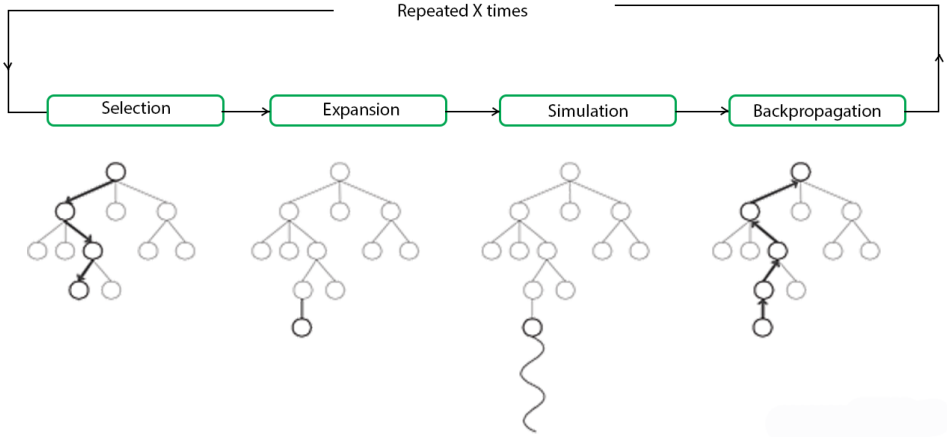
Principle of Operation

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.

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.

Principle of Operation

Each round of Monte Carlo tree search consists of four steps:



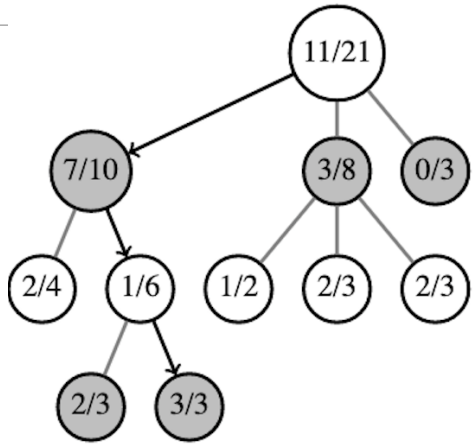
Selection

A selection function is applied recursively until a leaf node is reached.

The function attempts to balance exploitation and exploration.

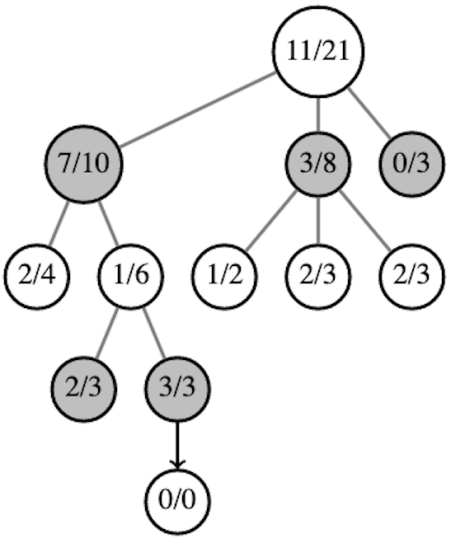
Exploitation: Select moves that lead to the best results so far

Exploration: Select less promising moves to remove uncertainty of them.



Expansion

One or more moves/states are created.

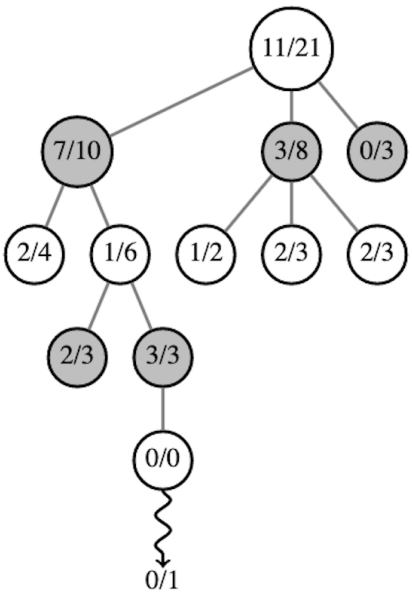


Simulation

One simulation game is played.

Actions are selected at random until game ends.

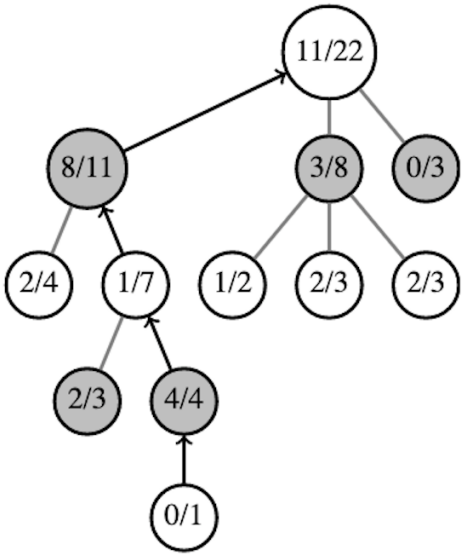
Use heuristic knowledge to bias selection to actions that look more promising.



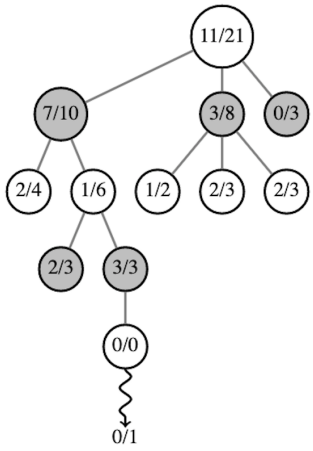
Backpropagation

After reaching the end of the simulated game, update each tree node that was traversed during that game.

The visit counts are increased and the win/loss ratio is modified according to the outcome.



Backpropagation

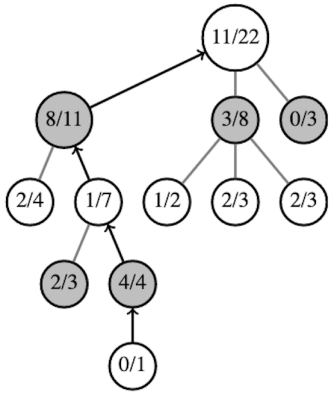


Each node shows the ratio of wins to total playouts from that point in the game tree for the player that node represents.

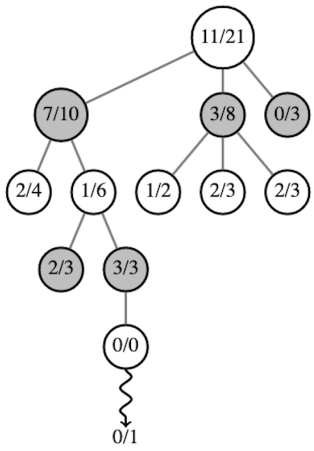
Black is about to move.

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.



Backpropagation



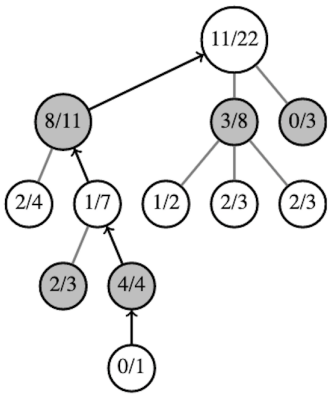
All nodes along selection path increase their simulation count (denominator)

If black wins, increment the win counts of all black nodes along the selection path (numerator)

If white wins, increment the win counts of all white nodes

Rounds of search are repeated as long as the time allotted to a move remains.

The move with the most simulations made (i.e. the highest denominator) is chosen as the final answer.



Putting it all together