Distributed Monte Carlo Tree Search

Introduction

Monte Carlo Tree Search(MCTS) is a method for finding optimal decisions in a given domain by taking random samples in the decision space and building a search tree according to the results.

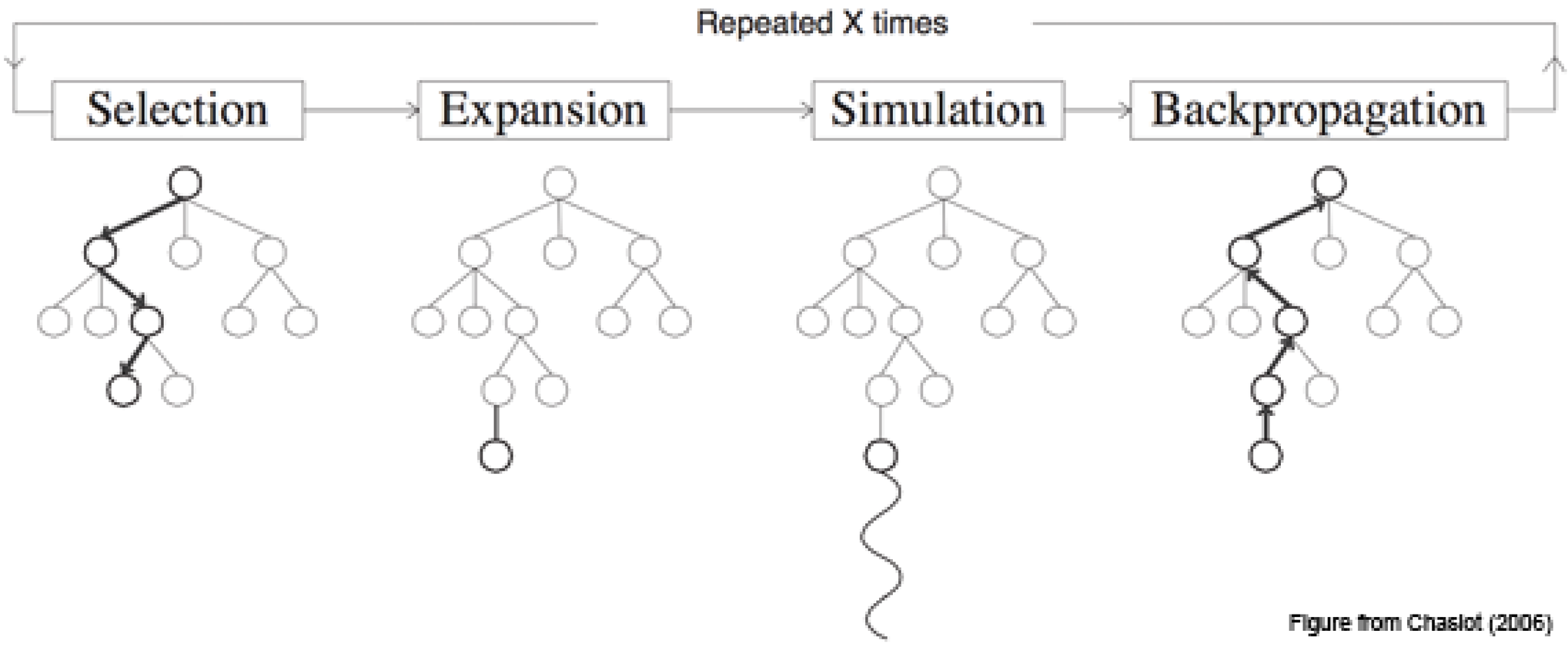


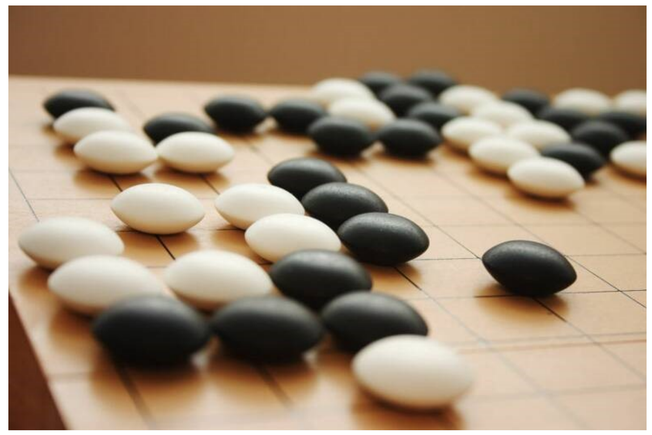
Figure from Chaslot (2006)

Applications

- *Board games:*
  - Hex
  - Go
  - Game of the Amazons

- *Real-time video games:*
  - Total War: Rome II

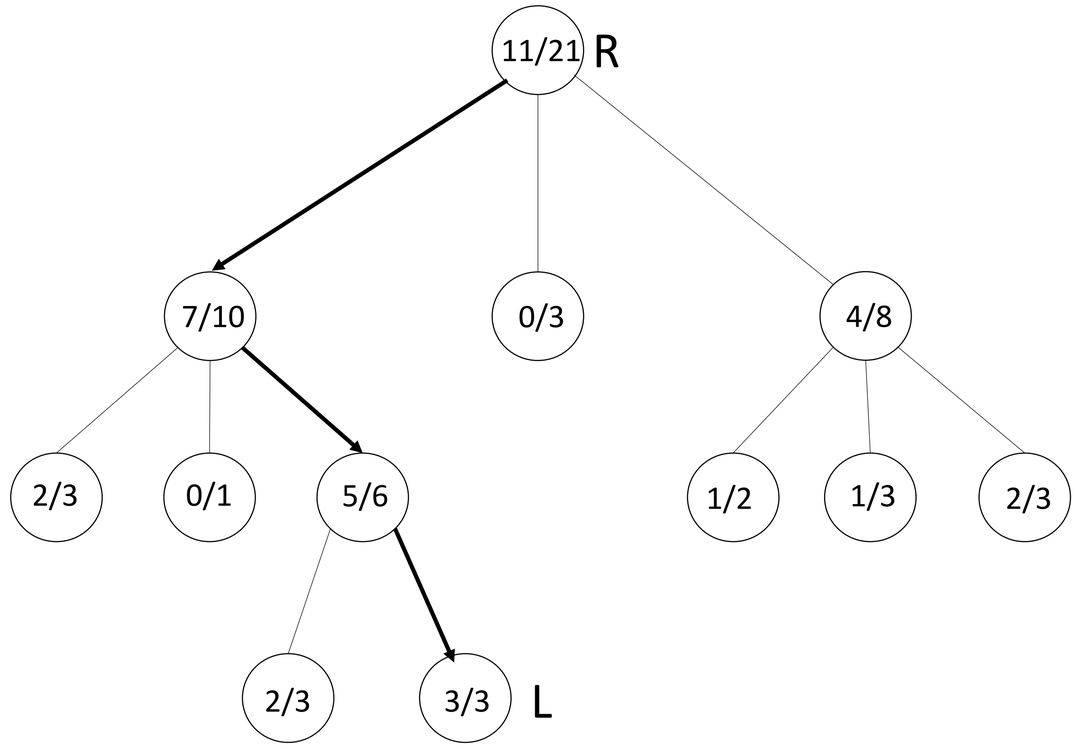
- *Nondeterministic games:*
  - poker
  - skat



Step 1: Selection

Begin with some root R, a tree policy is used to find the most urgent child of R, then we successively select child till we reach a leaf L.

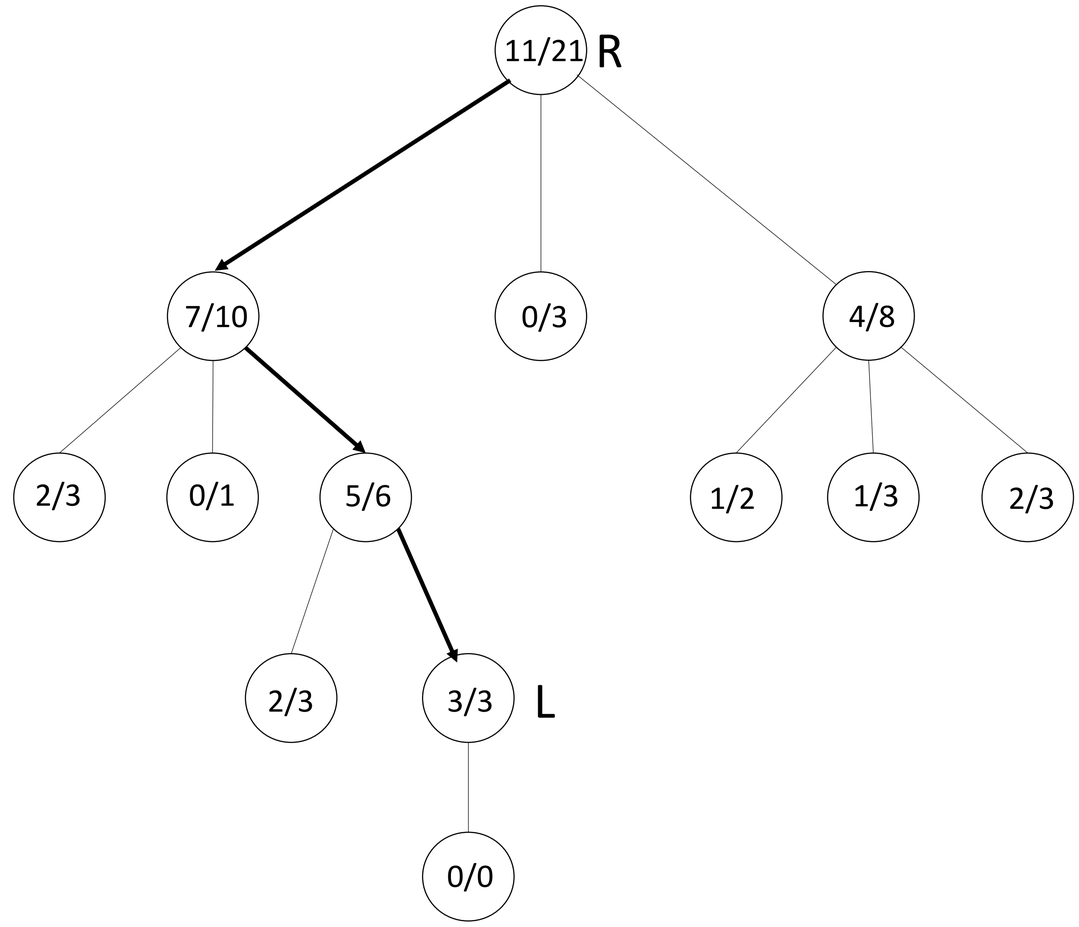
Each tree node stores the number of won/played playouts



Step 2: Expansion

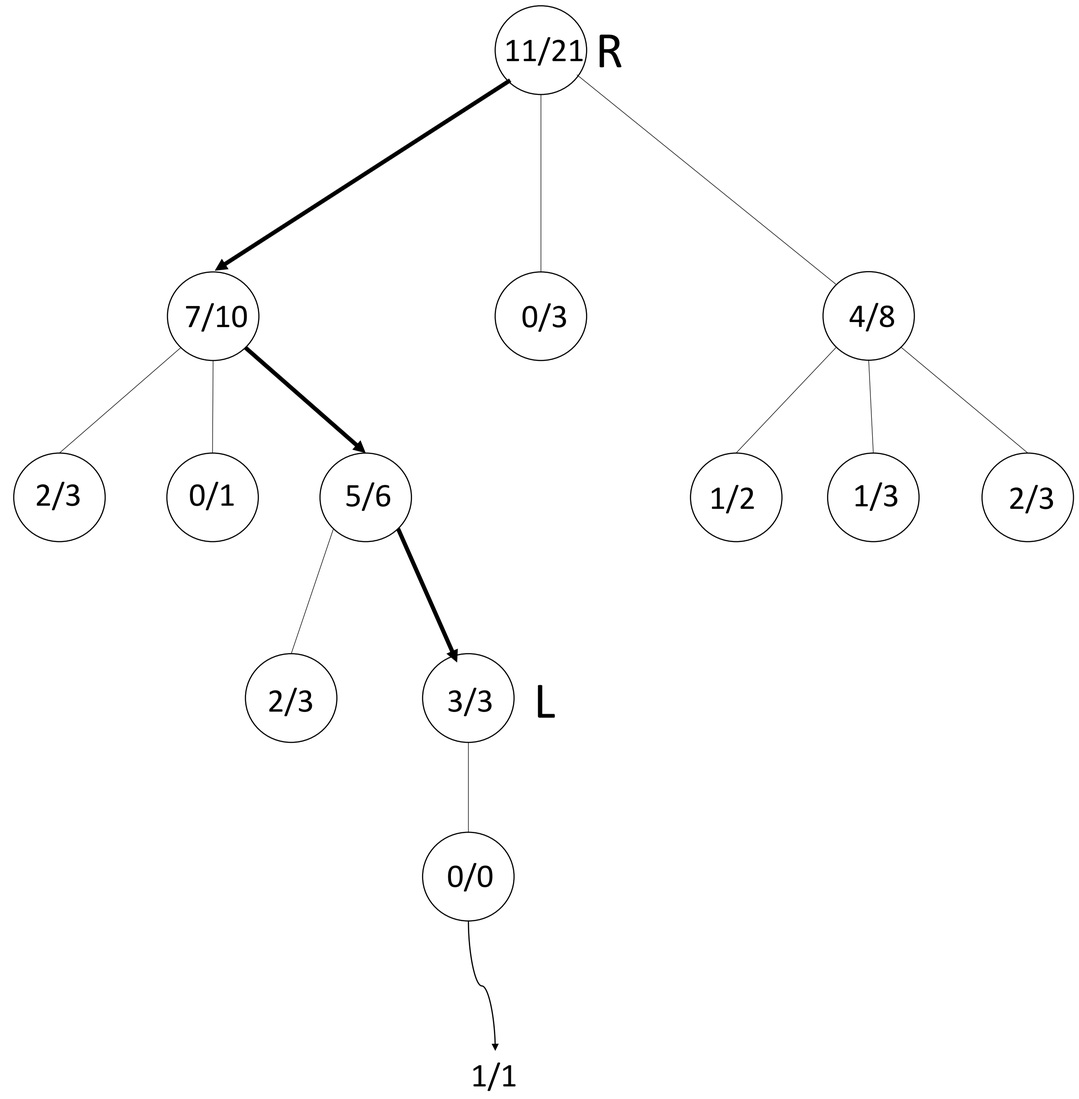
Unless the node L ends game, create one or more node of L and pick one of them, call it C.

Tree Policy $\frac{w_i}{n_i} + C\sqrt{\frac{\log t}{n_i}}$



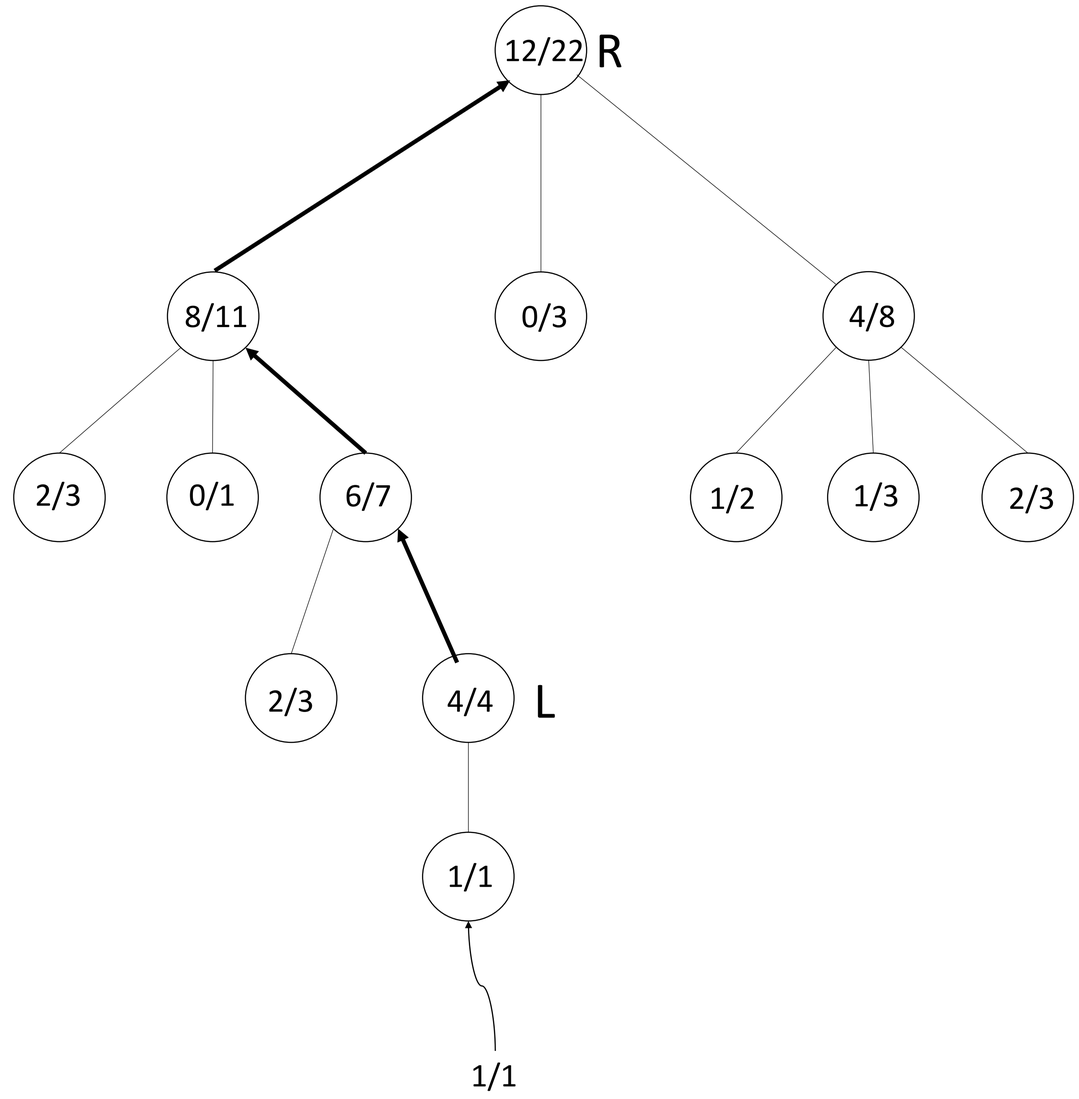
Step 3: Simulation

Simulate random playouts from C.



Step 4: Backpropagation

Update the information of the nodes in the path from C to R using the result of the random playouts.



Version 1

- *Modify the Expansion Stage*:

After we have selected a node to expand, we expand the node into *m* random children rather than a single child

- *Multiple Simulation*:

We simulate each child *k* times to get a better estimate of its node value.

Version 1

- *Advantages*:
  1. Able to simulate more moves per iteration
  2. More reliable estimates of interior nodes

- *Disadvantages*:
  1. After running tests, we found on larger sized problems simulating more moves from a given state does not improve game playing strength.
  2. Estimates of interior nodes are not very accurate no matter how many simulations we run if the graph is very tall

Version 2

- *Modify the Tree Policy*:

We no longer choose a node to expand deterministically, we now choose a node to expand randomly in proportion to its UCT value. *M* is now the number of such nodes chosen

- *New Tree Policy -> Select More Nodes*:

With this tree policy, we can run the selection algorithm and pick out different nodes to expand every time. This allows us to search not just the best move, but also other good moves.

- *Give unvisited nodes a UCT value*:

We give nodes that have yet to be visited a small UCT value so that in each iteration, we can explore past a single layer of children.

- *Modify k*:

Choose *k* to increase and decrease with *m*

Version 2

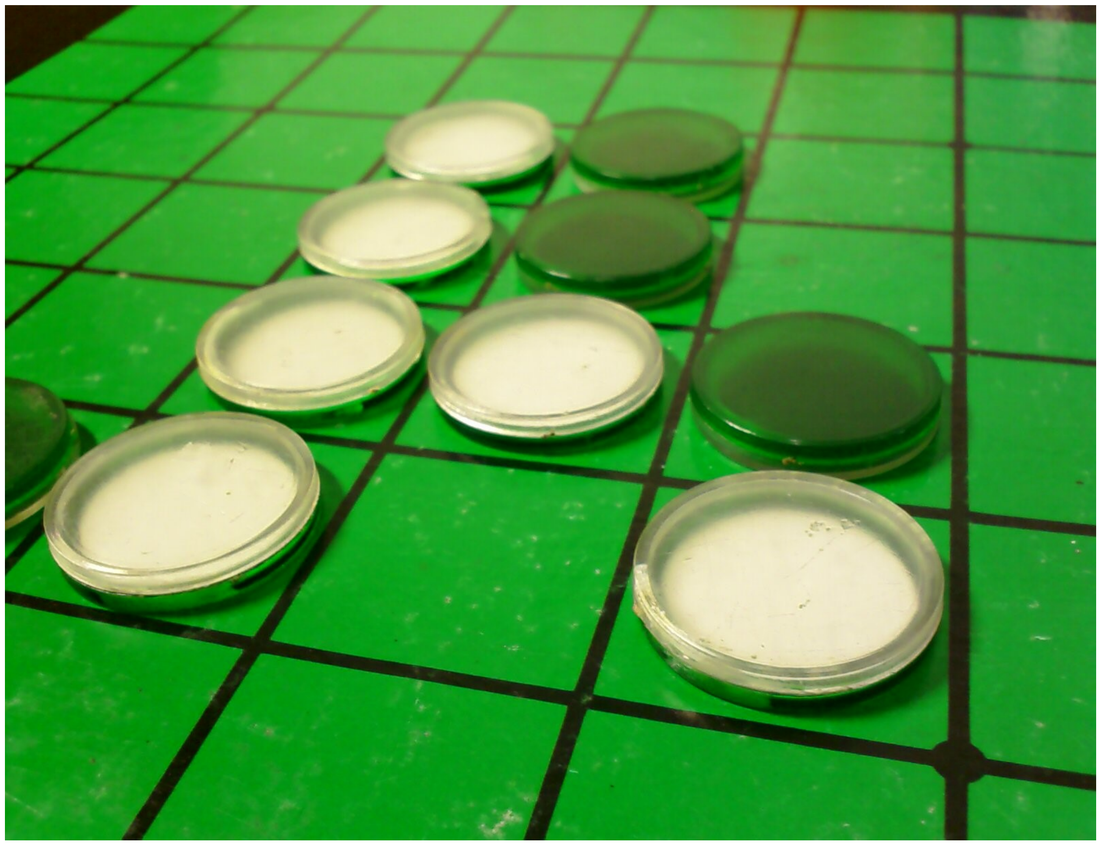
- *Advantages*:
  1. Explore moves apart form just the best move.
  2. Able to simulate deeper into the tree per iteration
- *Disadvantages*:
  What should the values of *m*, and *k* be?

Implementation - Othello

We apply MCTS to Othello in order to compare game strength of regular MCTS and distributed MCTS.

- *Othello*:

Each node in the tree is the state of the game, which is simply the board and the current player. The edges of the tree are the moves taken to go from state to state.



- *Map Reduce*:

The nodes of the graph chosen for simulation are mapped to the cluster and the results are collected as a list at the driver. The driver then does the backpropagation for each state locally.

Results

- *4x4 Board*: Distributed MCTS set to *m = 100, k = 10* and iterations = 10 vs. regular MCTS with iterations = 100 -> 24 games, 12 wins, 6 losses, 6 draws

- *6x6 Board*: Distributed MCTS set to *m = 100, k = 20* and iterations = 100 vs. regular MCTS with iterations = 1000 -> 10 games, 3 wins, 7 losses

- *6x6 Board*: Distributed MCTS set to *m = 200, k = 1* and iterations = 100 vs. regular MCTS with iterations = 1000 -> 10 games, 3 wins, 7 losses

- *6x6 Board*: Distributed MCTS set to *m = 100, k = 1* and iterations = 10 with custom UCT values vs. regular MCTS with iterations = 100 -> 10 games, 5 wins, 5 losses

*Both agents were given roughly the same amount of time to compute a move

- *Takeaway:*

Distributed MCTS struggles for larger board sizes since the problem size scales exponentially to board size, but *m does not.*

Analysis & Conclusion

- *Number of Iterations:*

Increasing the number of iterations in MCTS has a super-linear increase in game playing strength. However, distributed MCTS can not increase its iterations due its time dependence on the map reduce.

- *Time Dependent on # of Map Reduces:*

The majority of time in the algorithm is spent on the map reduce, which prevents us from increasing the number of iterations as a map reduce is quite time consuming.

- *Conclusion:*

We find that to match the strength of the regular MCTS we need a better tree policy that can work for the distributed MCTS.

Questions?