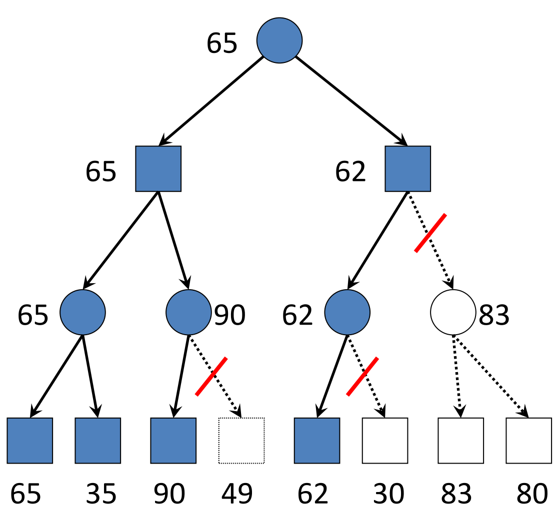
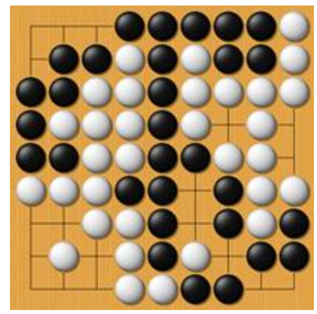
Introduction to Monte Carlo Methods and Monte Carlo Trees

Traditional Minimax



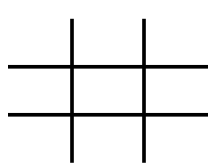
Minimax

- Doesn't work so well on games with very large branching factors/search space
  - Game of Go often cited
  - 9x9 here but normally 19x19
  - Can try heuristics for pruning
    - Don't seem to work that well



Monte Carlo Approach

- An alternative to minimax is a Monte Carlo approach
  - Simulate complete game with random moves and use the results to pick the best move
  - Consider tic tac toe and playing many random games, we would likely find that moving in the center first resulted in more wins than moving into one of the sides



Monte Carlo Example

- Applet for playing Connect-Four
- http://beej.us/blog/2010/01/monte-carlo-method-for-game-ai/

- Space requirements?
- Runtime?
- Heuristic?
- Cases where this fails?

One Solution

- Merge traditional minimax search with Monte Carlo approach
- Can do a minimax search to some depth then use Monte Carlo as an evaluation function
  - Requires ability to complete minimax to some reasonable depth (e.g. at least 2 or more ply)
- Other approaches attempt to balance how we explore the top part of the tree instead of deterministic like minimax

Background – Multi arm bandit problem

- Consider a slot machine with K arms
  - Pulling arms in sequences give different random payouts
  - What arms should you pull to maximize your payout given some number of coins to play?
- Dilemma: Explore or Exploit?
  - Explore: Test to find out the best arm
  - Exploit: Pull the best arm we have found so far to get some payout

Balancing Exploitation vs. Exploration

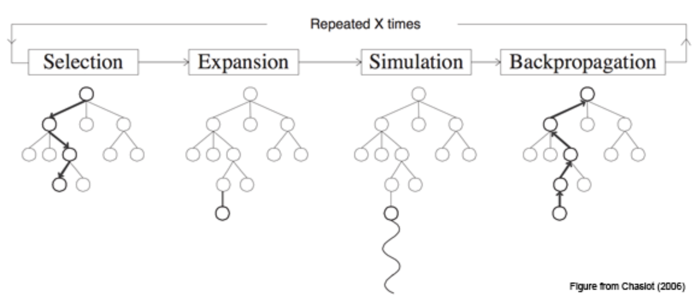
- Upper Confidence Bound
  - For arm i
    - $Payout_i$ = $ won playing arm i
    - n_i = Number of times arm i played
    - N = total number of plays so far
    - Can multiply exploration constant c in front of bias
- Pull the arm with the highest UCB
  - Expected payout rewards arm that has paid
  - Bias increases for arm that hasn't been played much
    - Maybe it's been unlucky and we need to try it again
  - There is theory that performance from the optimal is bounded

$$UCB_i = \underbrace{\frac{Payout_i}{n_i}}_{\text{Expected Payout}} + \underbrace{\sqrt{\frac{lgN}{n_i}}}_{\text{Bias}}$$

Applying to Trees

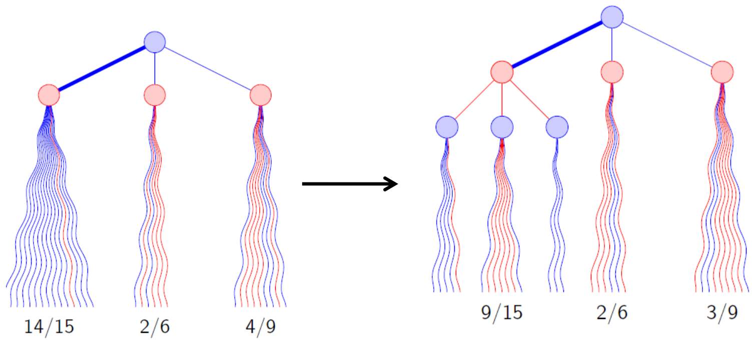
- Kocsis and Szepervari (2006) “Bandit based Monte-Carlo Planning”
  - Formalized a complete Monte Carlo Tree Search algorithm by extending UCB to minimax tree search
  - Named it the Upper Confidence Bounds for Trees (UCT) method
  - Most MCTS algorithms use UCT method

Basic MCTS Algorithm



Selection: Recursively pick best node that maximizes UCB for Trees (UCT) as long as the node is visited more than N_0 times
Expansion: Add child node(s) off the selected node to the list of possible nodes we can select in the next round; only 1 node in simplest implementation
Simulation: Randomly simulate game to completion
Backprop: Update nodes on the path with simulation results (wins, number of visits)

MCTS Visualization



- No minimax backup; only backup the outcomes to compute UCT
- Proven to converge to the minimax value
- Explores tree in a best-first manner

Success of Monte Carlo Tree Search

- Considered a breakthrough for Go
  - Used by best programs able to beat amateur humans
- Doesn't require a heuristic and can be used for problems with large branching factors
- Other gaming applications; good where there is randomness or uncertainty
  - Settlers of Catan
  - Real Time Strategy Games
  - Can still be used with classical board games
  - Might work well for TZAAR?
- Workshops devoted to MCTS

Resources

- http://www.mcts.ai
- http://remi.coulom.free.fr/Hakone2007/