Game Tree Extensions

Random Outcomes

Incomplete Information

Simultaneous Moves
Randomness: Moves by nature

Compute expected values:

- (-11 * .4) + (8 * .6) = 0.4
- (73 * .4) + (24 * .6) = 43.6
Incomplete Information

Information Set:
A set of decision nodes among which a player cannot distinguish.
Optimal play may require randomizing your action to avoid predictability.

Key idea: mixed strategy Nash equilibrium
Monte Carlo Tree Search

2/16/18
General Game Playing

Suppose you wanted to write an AI that could play any game. How could you do that?

First idea: backward induction

• Plays optimally once the full tree is explored.
• Exploring the whole tree may not be feasible.

Second idea: generalized heuristics

• When the tree is too big, we use depth-bounded search.
• We had to write new versions of basicEval and betterEval for each game.
• How can we make a heuristic that works for any game?
Monte Carlo Simulation Heuristic

Key idea: play out a bunch of random games.

To evaluate a state:
- Play random moves until the end of the game.
- Record the result.
- Repeat a bunch of times.
- Return the average result over the random games.
Monte Carlo Heuristic Pseudocode

function MC_BoardEval(state):
    wins = losses = draws = 0
    for i in 1:NUM_SAMPLES:
        next_state = state
        while non_terminal(next_state):
            next_state = makeRandomMove(next_state)
            incrementwins/losses/draws
    return (wins + .5*draws) / NUM_SAMPLES

Here, I’m using the following utility function:

• win: 1 point, draw: ½ points, loss: 0 points
• Many other other utility functions are possible.
Monte Carlo Board Evaluation

**Advantages**

- simple
- domain independent
- anytime algorithm

**Disadvantages**

- slow
- non-static
- nondeterministic
Improving MC_BoardEval

If we do min/max search with a depth limit of 4 and use MC_BoardEval as our heuristic. What happens at depth 3?

D=3

- p1 wins: 103
  p2 wins: 717
  draws: 180
  $u_1 = 0.1930$

D=4

- p1 wins: 723
  p2 wins: 156
  draws: 121
  $u_1 = 0.7835$
- p1 wins: 65
  p2 wins: 662
  draws: 273
  $u_1 = 0.2015$
- p1 wins: 725
  p2 wins: 162
  draws: 113
  $u_1 = 0.7815$
- p1 wins: 228
  p2 wins: 472
  draws: 300
  $u_1 = 0.378$

really bad
really close
Improving MC_BoardEval

• We’d like to do more simulations from the more-important states (those more likely to be picked).
• We’re willing to do fewer simulations from the less important (low value) states.

Key idea: explore/exploit tradeoff
• Explore: try something new
• Exploit: try something you know is good
Multi-armed bandit problem

It turns out we’ve stumbled upon a well-studied problem.

Given a row of slot machines (bandits), with different, unknown, probabilities of winning a jackpot, use a fixed number of quarters to win as many jackpots as possible.
Upper confidence bound (UCB)

Provably optimal solution to multi-armed bandit.

Pick nodes with probability proportional to:

\[ v_i + C \times \sqrt{\frac{\ln(N)}{n_i}} \]

- probability decreases in number of visits (explore)
- probability increases in a node’s value (exploit)
- always tries every option once
Why do UCB at only one level?

1. Why might we want to do UCB (instead of Min/Max) at shallower levels?
2. Why might we want to do UCB (instead of uniform random moves) at deeper levels?
Doing UCB at more levels

Extend to deeper levels?
+ more value out of every random playout
− more information to keep track of

How can we alleviate this?

Extend to shallower levels?
+ guide the search to explore better paths first
− lose optimality of minimax

Is this a big deal?
− never completely prune branches

Is this a big deal?