Monte Carlo Tree Search

2-15-16

Reading Quiz

What is the relationship between Monte Carlo tree search and upper confidence bound applied to trees?

a) MCTS is a type of UCB
b) UCB is a type of MCTS
c) both (they are the same algorithm)
d) neither (they are different algorithms)

Consider hex on an NxN board.

branching factor ≤ N^2

$2N \leq$ depth $\leq N^2$

board size	max branching factor	min depth	tree size	depth of 10^{10} nodes
6x6	36	12	$>10^{17}$	7
8x8	64	16	$>10^{28}$	6
11x11	121	22	$>10^{44}$	5
19x19	361	38	$>10^{96}$	4

Heuristics are hard.

Think about your board evaluation heuristics for Hex.

- Lots of human effort goes into designing a good heuristic.
- That effort isn’t transferrable to other domains.

Monte Carlo simulations

Idea: evaluate states by playing out random games.

```
function MC_BoardEval(state):
    wins = 0
    losses = 0
    for i=1:NUM_SAMPLES
        next_state = state
        while non_terminal(next_state):
            next_state = random_legal_move(next_state)
        if next_state.winner == state.turn: wins++
        else: losses++ #needs slight modification if draws possible
    return (wins - losses) / (wins + losses)
```

Monte Carlo board evaluation

Advantages

- simple
- domain independent
- anytime

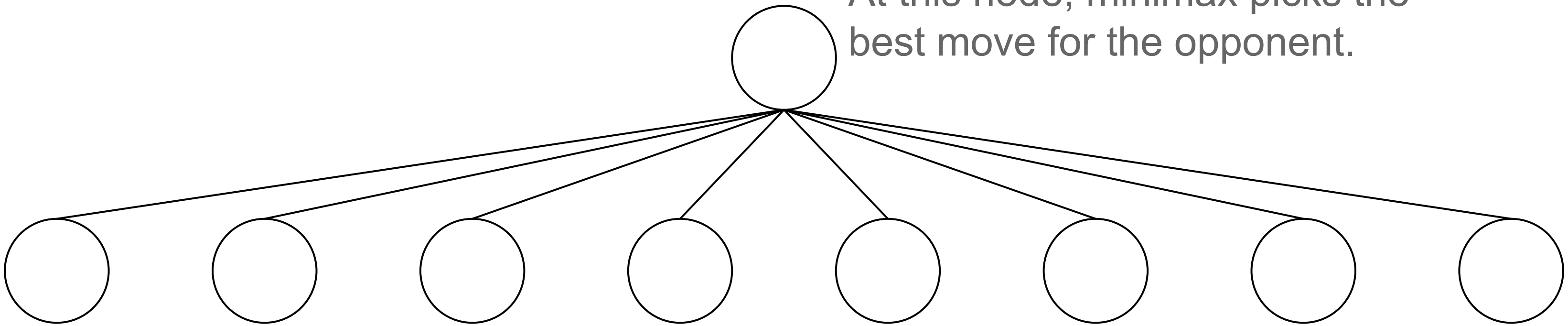
Disadvantages

- slow
- nondeterministic
- not great for alpha-beta pruning

Improving MC_BoardEval

Consider one level up. Suppose we're doing minimax search with a depth limit of 4 and using MC_BoardEval as our heuristic. What's happening at depth 3?

At this node, minimax picks the best move for the opponent.

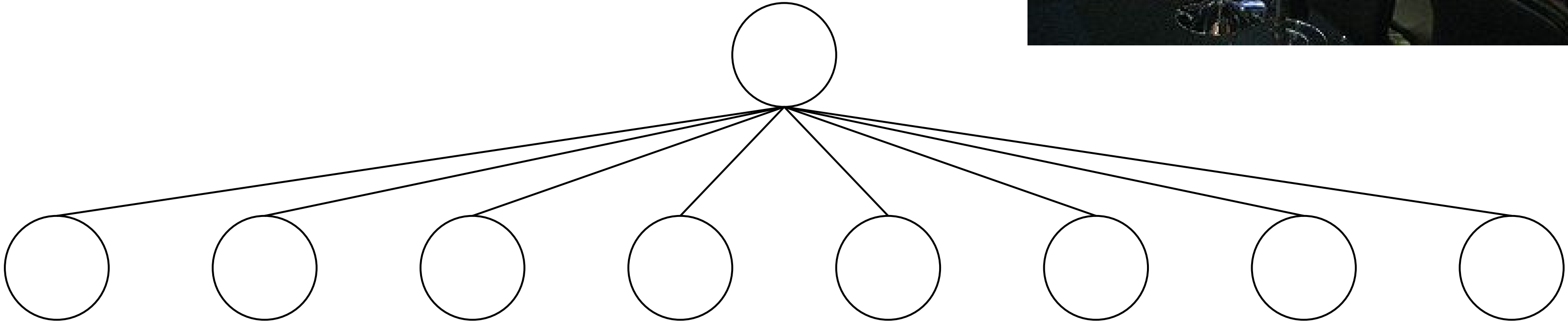


MC_BoardEval plays out NUM_SAMPLES random games from each of these nodes.

Objective: allocate samples more effectively.

Multi-armed bandit 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)

Pick each node with probability proportional to:

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

parent node visits

value estimate

tunable parameter

number of visits

- probability is decreasing in the number of visits (explore)
- probability is increasing in a node’s value (exploit)
- always tries every option once

Why do this at only one level?

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?)

The Monte Carlo tree search algorithm

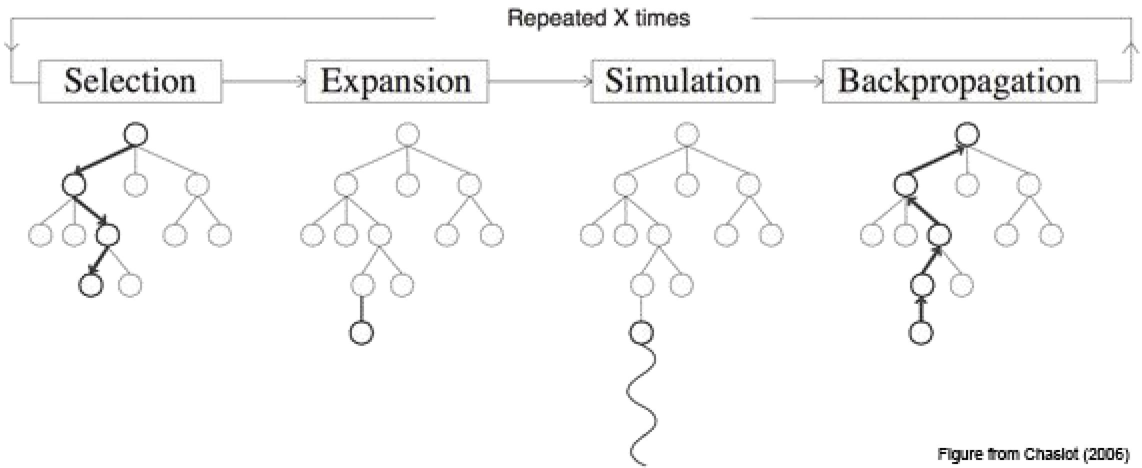


Figure from Chaslot (2006)

Selection

- Used for nodes we’ve seen before.
- Pick according to UCB.

Expansion

- Used when we reach the frontier.
- Add one node per playout.

Simulation

- Used beyond the search frontier.
- Don’t bother with UCB, just play randomly.

Backpropagation

- After reaching a terminal node.
- Update value and visits for states expanded in selection and expansion.

Basic MCTS pseudocode

```
function MCTS_sample(state)
    state.visits++
    if all children of state expanded:
        next_state = UCB_sample(state)
        winner = MCTS_sample(next_state)
    else:
        if some children of state expanded:
            next_state = expand(random unexpanded child)
        else:
            next_state = state
        winner = random_playout(next_state)
    update_value(state, winner)
```

MCTS helper functions

```
function UCB_sample(state):
    weights = []
    for child of state:
        w = child.value + C * sqrt(ln(state.visits) / child.visits)
        weights.append(w)
    distribution = [w / sum(weights) for w in weights]
    return child sampled according to distribution

function random_playout(state):
    if is_terminal(state):
        return winner
    else: return random_playout(random_move(state))
```

MCTS helper functions

```
function expand(state):
    state.visits = 1
    state.value = 0


function update_value(state, winner):
    # Depends on the application. The following would work for hex.
    if winner == state.turn:
        state.value += 1
    else:
        state.value -= 1
```

Note: reading assignments

- Wednesday has been updated to include sections 3.2-3.3.
- Friday has been updated to include miscellaneous short sections.
- Next week’s reading may change. I’ll send out an email if it does.