Extending MCTS

2-17-16
Reading Quiz (from Monday)

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

a) MCTS is a type of UCT

b) UCT is a type of MCTS

c) both (they are the same algorithm)

d) neither (they are different algorithms)
Reading Quiz

Which of these functions from the lab4 pseudocode implements the **tree policy**?

a) UCB_sample

b) random_playout

c) backpropagation

d) none of these
Generic MCTS algorithm

The **tree policy** returns a child node in the explored region of the tree.

UCT’s **tree policy** draws samples according to UCB.

The **default policy** returns a value estimate for a newly expanded node.

UCT’s **default policy** completes a uniform random playout.
function MCTS(root, rollouts)
    for i = 1 : rollouts
        node = root
        # selection
        while all children expanded and node is not terminal
            node = UCB_sample(node)
        # expansion
        if node not terminal
            node = expand(random unexpanded child of node)
        # simulation
        outcome = random_playout(node's state)
        # backpropagation
        backpropagation(node, root, outcome)
    return move that generates the highest-value successor of root
    (from the current player's perspective)
function UCB_sample(node)
    weights = [UCB_weight(child) for each child of node]
    distribution = normalize(weights)
    return random sample from distribution

function random_playout(state)
    while state is not terminal
        state = random successor of state
    return winner

function backpropagation(node, root, outcome):
    until node is root
        increment node's visits
        update_value(node, outcome)
    node = parent of node
Upper confidence bound (UCB)

Pick each node with probability proportional to:

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

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

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

\[ C = 2 \]

\[
\begin{align*}
\text{visits} & = 19 \\
\text{value} & = .68 \\
\text{visits} & = 5 \\
\text{value} & = .6 \\
\text{visits} & = 2 \\
\text{value} & = .5 \\
\text{visits} & = 12 \\
\text{value} & = .75 \\
\text{visits} & = 1 \\
\text{value} & = 0
\end{align*}
\]

\[
\begin{align*}
w & = [ 2.13 \quad 2.93 \quad 1.74 \quad 3.43 ] \\
\text{prob} & = [ .209 \quad .286 \quad .170 \quad .335 ]
\end{align*}
\]
The next time we select the parent...

Which values change?

How much?

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

\[ C = 2 \]

visits = 5
value = .6

visits = 2
value = .5

visits = 12
value = .75

visits = 2
value = 0

\[ w = [ 2.15, 2.95, 1.75, 2.45 ] \]

\[ \text{prob} = [ 0.209, 0.286, 0.170, 0.335 ] \]
Alternative tree policies

The tree policy must trade off exploration and exploitation.

- Epsilon-greedy: pick a uniform random child with probability $\varepsilon$ and the best child with probability $(1-\varepsilon)$.
- Use UCB, but seed the tree within initial values.
  - from previous runs
  - based on a heuristic
- Other ideas?
Alternative default policies

The default policy must be fast to evaluate and return a value estimate.

- Use the board evaluation heuristic from bounded minimax.
- Run multiple random rollouts for each expanded node.
- Other ideas?
Options for returning a move

- Return the neighbor with the best value estimate.
- Return the neighbor you’ve visited the most.
- Some combination of the above:
  - Continue simulating until they agree.
  - Use some weighted combination.
  - Question: could we use UCB_weight for this?
Extension: dynamic or unobservable environment

We’re already doing Monte Carlo sampling; just sample over the unknowns!

When we select this action, go to the left child 40% of the time and the right child 60%.
Extension: non-zero-sum games

- We now have a tuple of utilities at each outcome node.
- We can maintain a tuple of value estimates at each search tree node.
- The agent deciding at the parent node will use its entry in the value tuple when picking a child node to expand.
Exercise: construct the UCB distribution

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

\[ C = 2 \]

<table>
<thead>
<tr>
<th>Visits</th>
<th>Value</th>
<th>( w )</th>
<th>prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>(2.4, 3.4, 2.55)</td>
<td>4.55</td>
<td>.234</td>
</tr>
<tr>
<td>5</td>
<td>(0, 3, 5)</td>
<td>3.45</td>
<td>.177</td>
</tr>
<tr>
<td>12</td>
<td>(2, 4, 1)</td>
<td>5.00</td>
<td>.257</td>
</tr>
<tr>
<td>1</td>
<td>(6, 3, 4)</td>
<td>6.46</td>
<td>.332</td>
</tr>
</tbody>
</table>
## Comparing to minimax / backwards induction

<table>
<thead>
<tr>
<th>UCT / MCTS</th>
<th>Minimax / Backwards Induction</th>
</tr>
</thead>
<tbody>
<tr>
<td>• optimal with infinite rollouts</td>
<td>• optimal once the entire tree is explored or pruned</td>
</tr>
<tr>
<td>• anytime algorithm (can give an answer immediately, improves its answer with more time)</td>
<td>• can prove the outcome of the game</td>
</tr>
<tr>
<td>• A heuristic is not required, but can be used if available.</td>
<td>• Can be made anytime-ish with iterative deepening.</td>
</tr>
<tr>
<td>• Handles incomplete information gracefully.</td>
<td>• A heuristic is required unless the game tree is small.</td>
</tr>
<tr>
<td></td>
<td>• Hard to use on incomplete information games.</td>
</tr>
</tbody>
</table>