CSEP 573: Artificial Intelligence

ExpectiMax – Complex Games

slides adapted from Stuart Russel, Dan Klein, Pieter Abbeel from ai.berkeley.edu
And Hanna Hajishirzi, Jared Moore, Dan Weld
Video of Demo Min vs. Exp (Min)
Worst-Case vs. Average Case

Idea: Uncertain outcomes controlled by chance, not an adversary!
Expectimax Search

- Why wouldn’t we know what the result of an action will be?
  - Explicit randomness: rolling dice
  - Unpredictable opponents: the ghosts respond randomly
  - Unpredictable humans: humans are not perfect
  - Actions can fail: when moving a robot, wheels might slip

- Values should now reflect average-case (expectimax) outcomes, not worst-case (minimax) outcomes

- Expectimax search: compute the average score under optimal play
  - Max nodes as in minimax search
  - Chance nodes are like min nodes but the outcome is uncertain
  - Calculate their expected utilities
  - I.e. take weighted average (expectation) of children

- Later, we’ll learn how to formalize the underlying uncertain-result problems as Markov Decision Processes
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v

v = (1/2) (8) + (1/3) (24) + (1/6) (-12) = 10
Minimax

function decision(s) returns an action
    return the action a in Actions(s) with the highest minimax_value(Result(s,a))

function minimax_value(s) returns a value
    if Terminal-Test(s) then return Utility(s)
    if Player(s) = MAX then return max_a in Actions(s) minimax_value(Result(s,a))
    if Player(s) = MIN then return min_a in Actions(s) minimax_value(Result(s,a))
Expectiminimax

function decision(s) returns an action
  return the action a in Actions(s) with the highest value(Result(s,a))

function value(s) returns a value
  if Terminal-Test(s) then return Utility(s)
  if Player(s) = MAX then return \( \max_a \) in Actions(s) value(Result(s,a))
  if Player(s) = MIN then return \( \min_a \) in Actions(s) value(Result(s,a))
  if Player(s) = CHANCE then return \( \sum_a \) in Actions(s) Pr(a) * value(Result(s,a))
Expectimax Example
Expectimax Pruning?
Depth-Limited Expectimax

- Limited Expectimax
- Estimate of true expectimax value (which would require a lot of work to compute)
What Probabilities to Use?

- In expectimax search, we have a probabilistic model of how the opponent (or environment) will behave in any state:
  - Model could be a simple uniform distribution (roll a die)
  - Model could be sophisticated and require a great deal of computation
  - We have a chance node for any outcome out of our control: opponent or environment
  - The model might say that adversarial actions are likely!

- For now, assume each chance node magically comes along with probabilities that specify the distribution over its outcomes.

Having a probabilistic belief about another agent’s action does not mean that the agent is flipping any coins!
Quiz: Informed Probabilities

Let’s say you know that your opponent is actually running a depth 2 minimax, using the result 80% of the time, and moving randomly otherwise.

Question: What tree search should you use?

Answer: Expectimax!

- To figure out EACH chance node’s probabilities, you have to run a simulation of your opponent.
- This kind of thing gets very slow very quickly.
- Even worse if you have to simulate your opponent simulating you...
- … except for minimax and maximax, which have the nice property that it all collapses into one game tree.
Modeling Assumptions
The Dangers of Optimism and Pessimism

Dangerous Optimism
Assuming chance when the world is adversarial

Dangerous Pessimism
Assuming the worst case when it’s not likely
Assumptions vs. Reality

Pacman used depth 4 search with an eval function that avoids trouble
Ghost used depth 2 search with an eval function that seeks Pacman

Results from playing 5 games:

<table>
<thead>
<tr>
<th></th>
<th>Adversarial Ghost</th>
<th>Random Ghost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimax Pacman</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Expectimax Pacman</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pacman won 5/5, Avg. Score: 483
Ghost won 5/5, Avg. Score: 493
Pacman won 1/5, Avg. Score: -303
Video of Demo World Assumptions
Random Ghost – Expectimax Pacman
Video of Demo World Assumptions
Adversarial Ghost – Minimax Pacman
Video of Demo World Assumptions
Random Ghost – Minimax Pacman

SCORE: -1
Video of Demo World Assumptions
Adversarial Ghost – Expectimax Pacman
Assumptions vs. Reality

Pacman used depth 4 search with an eval function that avoids trouble
Ghost used depth 2 search with an eval function that seeks Pacman

Results from playing 5 games

<table>
<thead>
<tr>
<th></th>
<th>Adversarial Ghost</th>
<th>Random Ghost</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Minimax Pacman</strong></td>
<td>Won 5/5</td>
<td>Won 5/5</td>
</tr>
<tr>
<td></td>
<td>Avg. Score: 483</td>
<td>Avg. Score: 493</td>
</tr>
<tr>
<td><strong>Expectimax Pacman</strong></td>
<td>Won 1/5</td>
<td>Won 5/5</td>
</tr>
<tr>
<td></td>
<td>Avg. Score: -303</td>
<td>Avg. Score: 503</td>
</tr>
</tbody>
</table>
Other Game Types
Mixed Layer Types

- E.g. Backgammon
- Expecti-minimax
  - Environment is an extra “random agent” player that moves after each min/max agent
  - Each node computes the appropriate combination of its children

if \textit{state} is a \texttt{MAX} node then
\begin{align*}
&\text{return the highest } \text{\texttt{EXPECTIMINIMAX-VALUE of SUCCESSORS}}(\textit{state}) \\
\end{align*}

if \textit{state} is a \texttt{MIN} node then
\begin{align*}
&\text{return the lowest } \text{\texttt{EXPECTIMINIMAX-VALUE of SUCCESSORS}}(\textit{state}) \\
\end{align*}

if \textit{state} is a chance node then
\begin{align*}
&\text{return average of } \text{\texttt{EXPECTIMINIMAX-VALUE of SUCCESSORS}}(\textit{state}) \\
\end{align*}
Example: Backgammon

- Dice rolls increase $b$: 21 possible rolls with 2 dice
  - Backgammon $\approx$ 20 legal moves
  - Depth 2 = $20 \times (21 \times 20)^3 = 1.2 \times 10^9$

- As depth increases, probability of reaching a given search node shrinks
  - So usefulness of search is diminished
  - So limiting depth is less damaging
  - But pruning is trickier...

- Historic AI: TDGammon uses depth-2 search + very good evaluation function + reinforcement learning: world-champion level play

- 1st AI world champion in any game!
Multi-Agent Utilities

- What if the game is not zero-sum, or has multiple players?

- Generalization of minimax:
  - Terminals have utility tuples
  - Node values are also utility tuples
  - Each player maximizes its own component
  - Can give rise to cooperation and competition dynamically...

```
1,6,6  7,1,2  6,1,2  7,2,1  5,1,7  1,5,2  7,7,1  5,2,5
```