CSE 573: Artificial Intelligence

Adversarial Search
Dan Weld

Based on slides from
Dan Klein, Stuart Russell, Pieter Abbeel, Andrew Moore and Luke Zettlemoyer
(best illustrations from ai.berkeley.edu)
Types of Environments

- Fully observable vs. partially observable
- Single agent vs. multi-agent
- Deterministic vs. stochastic
- Episodic vs. sequential
- Discrete vs. continuous

![Diagram showing the interaction between an agent and its environment, with arrows indicating the flow of signals: Sensors to Agent, and Environment to Actuators. The agent's actions are indicated by arrows going towards the environment.]
Types of Games

<table>
<thead>
<tr>
<th>Perfect Information</th>
<th>Deterministic</th>
<th>Imperfect Information</th>
<th>Chance</th>
</tr>
</thead>
<tbody>
<tr>
<td>chess, checkers, go, othello</td>
<td>backgammon, monopoly</td>
<td></td>
<td></td>
</tr>
<tr>
<td>stratego</td>
<td>bridge, poker, scrabble, nuclear war</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Number of Players? 1, 2, …?
Zero-Sum Games

- Agents have opposite utilities (values on outcomes)
- Lets us think of a single value that one maximizes and the other minimizes
- Adversarial, pure competition

General Games

- Agents have independent utilities (values on outcomes)
- Cooperation, indifference, competition, & more are possible
- More later on non-zero-sum games
Minimax Values

States Under Agent’s Control:

\[ V(s) = \max_{s' \in \text{successors}(s)} V(s') \]

States Under Opponent’s Control:

\[ V(s') = \min_{s \in \text{successors}(s')} V(s) \]

Terminal States:

\[ V(s) = \text{known} \]
Minimax Implementation

```python
def min_value(state):
    if leaf?(state), return U(state)
    initialize v = +\infty
    for each c in children(state):
        v = min(v, max_value(c))
    return v

def max_value(state):
    if leaf?(state), return U(state)
    initialize v = -\infty
    for each c in children(state):
        v = max(v, min_value(c))
    return v
```

\[
V(s) = \max_{s' \in \text{successors}(s)} V(s')
\]

\[
V(s') = \min_{s \in \text{successors}(s')} V(s)
\]

Need **Base case** for recursion

Slide adapted from Dan Klein & Pieter Abbeel - ai.berkeley.edu
Alpha-Beta Quiz

Max:

Min:

Search depth-first
Left to right
Order is important

Do all nodes matter?

Slide adapted from Dan Klein & Pieter Abbeel - ai.berkeley.edu
Stochastic Single-Player

- What if we don’t know what the result of an action will be? E.g.,
  - In solitaire, shuffle is unknown
  - In minesweeper, mine locations
- Can do **expectimax search**
  - Chance nodes, like actions except the environment controls the action chosen
  - Max nodes as before
  - Chance nodes take average (expectation) of value of children
ExpectiMax Search

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)… or more complex
- We have a node for every outcome out of our control: opponent or environment

For now, assume \( \forall \) states we magically have a distribution to assign probabilities to enemy-actions / environment outcomes
def value(s):
    if s is a max node return maxValue(s)
    if s is an exp node return expValue(s)
    if s is a terminal node return evaluation(s)

def maxValue(s):
    values = [value(s’) for s’ in successors(s)]
    return max(values)

def expValue(s):
    values = [value(s’) for s’ in successors(s)]
    weights = [probability(s, s’) for s’ in successors(s)]
    return expectation(values, weights)
ExpectiMax for Pacman

- Note: that we’ve gotten away from thinking that the ghosts are trying to minimize pacman’s score
- Instead, they are now a part of the environment
- Pacman has a belief (distribution) over how they will act

- Quiz: Can we see minimax as a special case of expectimax?
- Quiz: what would pacman’s computation look like if we assumed that the ghosts were doing 1-plex minimax and taking the result 80% of the time, otherwise moving randomly?
Expectimax for Pacman

Results from playing 5 games

<table>
<thead>
<tr>
<th></th>
<th>Minimizing Ghost</th>
<th>Random Ghost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimax Pacman</td>
<td>Won 5/5</td>
<td>Won 5/5</td>
</tr>
<tr>
<td></td>
<td>Avg. Score: 493</td>
<td>Avg. Score: 483</td>
</tr>
<tr>
<td>Expectimax Pacman</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>

Pacman does depth 4 search with an eval function that avoids trouble
Minimizing ghost does depth 2 search with an eval function that seeks Pacman
ExpectiMax Pruning?

- Not easy
  - exact: need bounds on possible values
  - approximate: sample high-probability branches
ExpectiMax Evaluation

- Evaluation functions quickly return an estimate for a node’s true value (which value, expectimax or minimax?)
- For minimax, evaluation function scale doesn’t matter
  - We just want better states to have higher evaluations (get the ordering right)
  - We call this insensitivity to monotonic transformations
- For expectimax, we need magnitudes to be meaningful
Mixed Layer Types

- E.g. Backgammon
- Expecti-Mini-Max
  - Environment is an extra player that moves after each agent
  - Chance nodes take expectations, otherwise like minimax

\[
\text{if } state \text{ is a Max node then return the highest } \text{ExpectiMinimax-Value of Successors}(state) \\
\text{if } state \text{ is a Min node then return the lowest } \text{ExpectiMinimax-Value of Successors}(state) \\
\text{if } state \text{ is a chance node then return average of } \text{ExpectiMinimax-Value of Successors}(state)
\]
Stochastic Two-Player

- Dice rolls increase $b$: 21 possible rolls with 2 dice
  - Backgammon $\approx$ 20 legal moves
  - Depth 4 = $20 \times (21 \times 20)^3 = 1.2 \times 10^9$

- As depth increases, probability of reaching a given node shrinks
  - So value of lookahead is diminished
  - So limiting depth is less damaging
  - But pruning is less possible…

- TDGammon used depth-2 search + very good eval function
  - Learned via NN & reinforcement learning
  - World-champion level play (1992, Gerald Tesauro)
Multi-player Non-Zero-Sum Games

Similar to minimax:

- Utilities are now tuples
- Each player maximizes their own entry at each node
- Propagate (or back up) nodes from children
- Can give rise to cooperation and competition dynamically…

In this example… three agents