ADVERSARIAL SEARCH

Lecture 3.2
CSE368: Artificial Intelligence
June 5, 2019

*Slides are adopted from:
• Prof. Nils Napp’s CSE368 course, Fall 2018
• UC Berkley course CS188 Introduction to AI, Spring 2019 (ai.berkeley.edu)
Adversarial search problems games

• They occur in multiagent competitive environments
• There is an opponent we can’t control planning again us!
• **Game vs. search**: optimal solution is not a sequence of actions but a strategy (policy) If opponent does a, agent does b, else if opponent does c, agent does d, etc.
• Tedious and fragile if hard-coded (i.e., implemented with rules)
• Good news: Games are modeled as search problems and use heuristic evaluation functions.
Adversarial Games

Many different kinds of games!

Axes:
- Deterministic or stochastic?
- One, two, or more players?
- Zero sum?
- Perfect information (can you see the state)?

Want algorithms for calculating a strategy (policy) which recommends a move from each state
### Type of Games

<table>
<thead>
<tr>
<th>Deterministic</th>
<th>Stochastic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perfect Information</td>
<td></td>
</tr>
<tr>
<td>Imperfect Information</td>
<td></td>
</tr>
</tbody>
</table>
## Type of Games

<table>
<thead>
<tr>
<th>Perfect Information</th>
<th>Deterministic</th>
<th>Chance</th>
</tr>
</thead>
<tbody>
<tr>
<td>chess, checkers, go, othello</td>
<td></td>
<td></td>
</tr>
<tr>
<td>backgammon monopoly</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Imperfect Information</td>
<td>battleships, blind tictactoe</td>
<td>bridge, poker, scrabble nuclear war</td>
</tr>
</tbody>
</table>
Embedded thinking or Backward reasoning

- One agent is trying to figure out what to do.
Embedded thinking or Backward reasoning

- One agent is trying to figure out what to do.
- How to decide? He thinks about the consequences of the possible actions.
Embedded thinking or Backward reasoning

- One agent is trying to figure out what to do.
- How to decide? He thinks about the consequences of the possible actions.
- He needs to think about his opponent as well
Embedded thinking or Backward reasoning

- One agent is trying to figure out what to do.
- How to decide? He thinks about the consequences of the possible actions.
- He needs to think about his opponent as well.
- The opponent is also thinking about what to do etc.
Embedded thinking or Backward reasoning

- One agent is trying to figure out what to do.
- How to decide? He thinks about the consequences of the possible actions.
- He needs to think about his opponent as well.
- The opponent is also thinking about what to do etc.
- Each will imagine what would be the response from the opponent to their actions.
Embedded thinking or Backward reasoning

- One agent is trying to figure out what to do.
- How to decide? He thinks about the consequences of the possible actions.
- He needs to think about his opponent as well.
- The opponent is also thinking about what to do etc.
- Each will imagine what would be the response from the opponent to their actions.
- This entails an embedded thinking.
Single player...
Adversarial search: minimax

- Two players: Max and Min
- Players alternate turns
- Max moves first
- Max maximizes results
- Min minimizes the result
- Compute each node’s minimax value’s the best achievable utility against an optimal adversary
Adversarial search: minimax

- computer’s turn
- opponent’s turn
- computer’s turn
- opponent’s turn
- leaf nodes are evaluated

The computer is Max. The opponent is Min.

At the leaf nodes, the utility function is employed. Big value means good, small is bad.
Adversarial search: minimax

Find the optimal strategy for Max:

- Depth-first search of the game tree
- An optimal leaf node could appear at any depth of the tree
- Minimax principle: compute the utility of being in a state assuming both players play optimally from there until the end of the game
- Propagate minimax values up the tree once terminal nodes are discovered
Adversarial search problems games

A game with 2 players (MAX and MIN, MAX moves first, turn-taking) can be defined as a search problem with:

- initial state: board position
- player: player to move
- successor function: a list of legal (move, state) pairs
- goal test: whether the game is over – terminal states
- utility function: gives a numeric value for the terminal states (win, loss, draw)

Game tree = initial state + legal moves
Deterministic Games

Many possible formalizations, one is:

**States:** $S$ (start at $s_0$)

**Players:** $P=\{1...N\}$ (usually take turns)

**Actions:** $A$ (may depend on player / state)

**Transition Function:** $S \times A \rightarrow S$

**Terminal Test:** $S \rightarrow \{t,f\}$

**Terminal Utilities:** $S \times P \rightarrow R$

Solution for a player is a **policy:** $S \rightarrow A$
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, and more are all possible
- More later on non-zero-sum games
Single-Agent Trees
Value of a state: The best achievable outcome (utility) from that state

Non-Terminal States:

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

\[ V(s) = \text{known} \]
Adversarial Game Trees

-20 -8 ... -18 -5 ... -10 +4 -20 +8
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:

\[ V(s) = \text{known} \]
Tic-Tac-Toe Game Tree
Adversarial Search (Minimax)

**DETERMINISTIC, ZERO-SUM GAMES:**
- Tic-tac-toe, chess, checkers
- One player maximizes result
- The other minimizes result

**MINIMAX SEARCH:**
- A state-space search tree
- Players alternate turns
- Compute each node’s **minimax value:**
  - the best achievable utility against a rational (optimal) adversary
Minimax Implementation

**Minimax Implementation**

```python
def min_value(state):
    initialize v = +\infty
    for each successor of state:
        v = min(v, max_value(successor))
    return v

def max_value(state):
    initialize v = -\infty
    for each successor of state:
        v = max(v, min_value(successor))
    return v
```

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

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

```python
def value(state):
    if the state is a terminal state: return the state’s utility
    if the next agent is MAX: return max-value(state)
    if the next agent is MIN: return min-value(state)

def max-value(state):
    initialize v = -\infty
    for each successor of state:
        v = max(v, value(successor))
    return v

def min-value(state):
    initialize v = +\infty
    for each successor of state:
        v = min(v, value(successor))
    return v
```
Minimax Example

```

```

```
Game Tree Pruning
Minimax Pruning
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
   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
Reminder: Probabilities

A **random variable** represents an event whose outcome is unknown.

A **probability distribution** is an assignment of weights to outcomes.

**Example: Traffic on freeway**
- Random variable: $T =$ whether there’s traffic
- Outcomes: $T$ in \{none, light, heavy\}
- Distribution: $P(T=\text{none}) = 0.25$, $P(T=\text{light}) = 0.50$, $P(T=\text{heavy}) = 0.25$

**Some laws of probability:**
- Probabilities are always non-negative.
- Probabilities over all possible outcomes sum to one.
Reminder: Expectations

The expected value of a function of a random variable is the average, weighted by the probability distribution over outcomes.

Example: How long to get to the airport?

Time: 20 min x 30 min x 60 min x 20 min + 30 min + 60 min = 35 min
Probability: 0.25 0.50 0.25
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!