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)
Outline

- Adversarial Search
  - Minimax search
  - $\alpha$-$\beta$ search
  - Evaluation functions
  - Expectimax

- Reminder:
  - Project 2 due in 7 days
Types of Environments

- Fully observable vs. partially observable
- Single agent vs. multi-agent
- Deterministic vs. stochastic
- Episodic vs. sequential
- Discrete vs. continuous
1994: Checkers. Chinook ended 40-year-reign of human world champion Marion Tinsley. Used search plus an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions. Checkers is now solved!
**Game Playing State-of-the-Art**

**1997: Chess.** Deep Blue defeated human world champion Gary Kasparov in a six-game match. Deep Blue examined 200 million positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.
Game Playing State-of-the-Art

**Go:** \( b > 300! \) Programs use monte carlo tree search + pattern KBs

2015: AlphaGo beats European Go champion Fan Hui (2 dan) 5-0

2016: AlphaGo beats Lee Sedol (9 dan) 4-1
Othello: Human champions refuse to compete against computers.
Game Playing State-of-the-Art

- Pacman: ... unknown ...
### Types of Games

<table>
<thead>
<tr>
<th>Deterministic</th>
<th>Chance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perfect Info.</td>
<td>Chess, Checkers, Go, Othello</td>
</tr>
<tr>
<td>Imperfect Info.</td>
<td>Stratego</td>
</tr>
<tr>
<td></td>
<td>Backgammon, Monopoly</td>
</tr>
<tr>
<td></td>
<td>Bridge, Poker, Scrabble, Nuclear War</td>
</tr>
</tbody>
</table>

Number of Players? 1, 2, …?
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, & more are possible
- More later on non-zero-sum games
Deterministic Single-Player

- Deterministic, single player, perfect information:
  - Know the rules, action effects, winning states
  - E.g. Freecell, 8-Puzzle, Rubik’s cube
- … it’s just search!
- Slight reinterpretation:
  - Each node stores a value: the best outcome it can reach
  - This is the maximal outcome of its children (the **max value**)
  - Note that we don’t have path sums as before (utilities at end)
- After search, can pick move that leads to best node
Deterministic Two-Player

- E.g. tic-tac-toe, chess, checkers
- Zero-sum games
  - One player maximizes result
  - The other minimizes result
Deterministic Two-Player

- E.g. tic-tac-toe, chess, checkers
- Zero-sum games
  - One player maximizes result
  - The other minimizes result

**Minimax search**
- A state-space search tree
- Players alternate
- Choose move to position with highest minimax value
  = best achievable utility against best play
Tic-tac-toe Game Tree

You choose

Opponent

You choose

Opponent

You choose

TERMINAL

Utility

-1

0

+1
Previously: Single-Agent Trees
Previously: Value of a State

Value of a state:
The best achievable outcome (utility) from that state

Non-Terminal States:

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

Terminal States:

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

Slide adapted from Dan Klein & Pieter Abbeel - ai.berkeley.edu
Adversarial Game Trees

Value(interior) ⇐ diffrent

Value(leaves) ⇐ same

Slide adapted from Dan Klein & Pieter Abbeel - ai.berkeley.edu
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:

$$V(s) = \text{known}$$

Slide adapted from Dan Klein & Pieter Abbeel - ai.berkeley.edu
Minimax Implementation

Need **Base case** for recursion

```python
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

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
```

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

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

Slide adapted from Dan Klein & Pieter Abbeel - ai.berkeley.edu
Concrete Minimax Example

max

min
Minimax Example

max

min

A_1

A_2

A_3

A_{11} A_{12} A_{13}

A_{21} A_{22} A_{23}

A_{31} A_{32} A_{33}

3 12 8

2 4 6

14 5 2
Quiz

Max:

Min:
Answer

Max:

Min:
Minimax Properties

- **Optimal?**
  - Yes, against perfect player. Otherwise?

- **Time complexity?**
  - $O(b^m)$

- **Space complexity?**
  - $O(bm)$

- **For chess, $b \sim 35$, $m \sim 100$**
  - Exact solution is completely infeasible
  - **But**, … do we need to explore the whole tree?
Do We Need to Evaluate Every Node?

Max:

Min:
Do We Need to Evaluate Every Node?

Max:

Min:

Progress of search...
\( \alpha-\beta \) Pruning Example

Max:

Min:

Progress of search...

Doesn't matter! Don't need to evaluate
Alpha-Beta Quiz

Search depth-first
Left to right
Order is important

Do all nodes matter?

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
\(\alpha - \beta\) Pruning

- \(\alpha\) is MAX’s best choice on path to root
- If \(n\) becomes worse than \(\alpha\), MAX will avoid it, so can stop considering \(n\)’s other children
- Define \(\beta\) similarly for MIN
def min-val(state):
    if leaf?(state), return U(state)
    initialize v = +\infty
    for each c in children(state):
        v = min(v, max-val(c))
    return v

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

Slide adapted from Dan Klein & Pieter Abbeel - ai.berkeley.edu
def min-val(state, α, β):
    if leaf?(state), return U(state)
    initialize v = +∞
    for each c in children(state):
        v = min(v, max-val(c, α, β))
    return v

def max-val(state, α, β):
    if leaf?(state), return U(state)
    initialize v = -∞
    for each c in children(state):
        v = max(v, min-val(c, α, β))
    return v
def min-val(state, α, β):
    if leaf?(state), return U(state)
    initialize v = +∞
    for each c in children(state):
        v = min(v, max-val(c, α, β))
        if v ≤ α return v
    β = min(β, v)
    return v

def max-val(state, α, β):
    if leaf?(state), return U(state)
    initialize v = -∞
    for each c in children(state):
        v = max(v, min-val(c, α, β))
        if v ≥ β return v
    α = max(α, v)
    return v

α: MAX's best option on path to root
β: MIN's best option on path to root

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

http://inst.eecs.berkeley.edu/~cs61b/fa14/ta-materials/apps/ab_tree_practice/
Alpha-Beta Pruning Properties

- This pruning has **no effect** on final result at the root

- **Values** of intermediate nodes might be wrong!
  - but, they are correct **bounds**

- Good child ordering improves effectiveness of pruning

- With “perfect ordering”:
  - Time complexity drops to $O(b^{m/2})$
  - **Doubles** solvable depth!
  - (But complete search of complex games, e.g. chess, is still hopeless…)**
Resource Limits

- **Problem:** In realistic games, cannot search to leaves!

- **Solution:** Depth-limited search
  - Instead, search only to a limited depth in the tree
  - Replace terminal utilities with an *evaluation function* for non-terminal positions

- **Example:**
  - Suppose we have 3 min/move, can explore 1M nodes/sec
  - So can check 200M nodes per move
  - $\alpha-\beta$ reaches about depth 10 $\rightarrow$ decent chess program

- Guarantee of optimal play is gone

- More plies makes a BIG difference
Depth Matters

- Evaluation functions are always imperfect
- The deeper in the tree the evaluation function is buried, the less the quality of the evaluation function matters
- Good example of the tradeoff between complexity of *features* and complexity of *computation*
Iterative deepening uses DFS as a subroutine:

1. Do a DFS which only searches for paths of length 1 or less. (DFS gives up on any path of length 2)
2. If “1” fails, do a DFS which only searches paths of length 2 or less.
3. If “2” fails, do a DFS which only searches paths of length 3 or less.

….and so on.

Can one adapt to games to make anytime algorithm?
Heuristic Evaluation Function

- Function which scores non-terminals

- Ideal function: returns the true utility of the position
  - In practice: need a simple, fast approximation
    - typically weighted linear sum of features:
    - e.g. \( f_1(s) = (\text{num white queens} - \text{num black queens}) \), etc.

\[
Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)
\]
Evaluation for Pacman

What features would be good for Pacman?

\[ \text{Eval}(s) = w_1f_1(s) + w_2f_2(s) + \ldots + w_nf_n(s) \]
Which algorithm?

α-β, depth 4, simple eval fun
Which algorithm?

$\alpha$-$\beta$, depth 4, better eval fun
Why Pacman Starves

- He knows his score will go up by eating the dot now.
- He knows his score will go up just as much by eating the dot later on.
- There are no point-scoring opportunities after eating the dot.
- Therefore, waiting seems just as good as eating.
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
Which Algorithms?

Expectimax

Minimax

3 ply look ahead, ghosts move randomly
Maximum Expected Utility

- Why should we average utilities? Why not minimax?

- Principle of maximum expected utility: an agent should choose the action which maximizes its expected utility, given its knowledge
  - General principle for decision making
  - Often taken as the definition of rationality
  - We’ll see this idea over and over in this course!

- Let’s decompress this definition…
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: $P(T=\text{none}) = 0.25$, $P(T=\text{light}) = 0.55$, $P(T=\text{heavy}) = 0.20$

- Some laws of probability (more later):
  - Probabilities are always non-negative
  - Probabilities over all possible outcomes sum to one

- As we get more evidence, probabilities may change:
  - $P(T=\text{heavy}) = 0.20$, $P(T=\text{heavy} \mid \text{Hour=8am}) = 0.60$
  - We’ll talk about methods for reasoning and updating probabilities later
What are Probabilities?

- Objectivist / frequentist answer:
  - Averages over repeated experiments
  - E.g. empirically estimating $P(\text{rain})$ from historical observation
  - E.g. pacman’s estimate of what the ghost will do, given what it has done in the past
  - Assertion about how future experiments will go (in the limit)
  - Makes one think of inherently random events, like rolling dice

- Subjectivist / Bayesian answer:
  - Degrees of belief about unobserved variables
  - E.g. an agent’s belief that it’s raining, given the temperature
  - E.g. pacman’s belief that the ghost will turn left, given the state
  - Often learn probabilities from past experiences (more later)
  - New evidence updates beliefs (more later)
Uncertainty Everywhere

- Not just for games of chance!
  - I’m sick: will I sneeze this minute?
  - Email contains “FREE!”: is it spam?
  - Tooth hurts: have cavity?
  - 60 min enough to get to the airport?
  - Robot rotated wheel three times, how far did it advance?
  - Safe to cross street? (Look both ways!)

- Sources of uncertainty in random variables:
  - Inherently random process (dice, etc)
  - Insufficient or weak evidence
  - Ignorance of underlying processes
  - Unmodeled variables
  - The world’s just noisy – it doesn’t behave according to plan!
Review: Expectations

- Real valued functions of random variables:

\[ f : X \rightarrow R \]

- Expectation of a function of a random variable

\[ E_{P(X)}[f(X)] = \sum_x f(x)P(x) \]

- Example: Expected value of a fair die roll

<table>
<thead>
<tr>
<th>( X )</th>
<th>P</th>
<th>( f )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1/6</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1/6</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1/6</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1/6</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>1/6</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>1/6</td>
<td>6</td>
</tr>
</tbody>
</table>

\[ 1 \times \frac{1}{6} + 2 \times \frac{1}{6} + 3 \times \frac{1}{6} + 4 \times \frac{1}{6} + 5 \times \frac{1}{6} + 6 \times \frac{1}{6} = 3.5 \]
Utilities

- Utilities are functions from outcomes (states of the world) to real numbers that describe an agent’s preferences.

- Where do utilities come from?
  - In a game, may be simple (+1/-1)
  - Utilities summarize the agent’s goals
  - Theorem: any set of preferences between outcomes can be summarized as a utility function (provided the preferences meet certain conditions).

- In general, we hard-wire utilities and let actions emerge (why don’t we let agents decide their own utilities?)

- More on utilities soon…