CSE 473: Artificial Intelligence  
Fall 2014

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
  - α-β search
  - Evaluation functions
  - Expectimax

- Reminder:
  - Project 1 due Monday

Game Playing State-of-the-Art

- Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used 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!

- Chess: Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997. 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.

- Othello: Human champions refuse to compete against computers, which are too good.

- Go: Human champions are beginning to be challenged by machines, though the best humans still beat the best machines on the full board. In go, b > 300, so need pattern knowledge bases and monte carlo search (UCT)

- Pacman: unknown

Types of Games

<table>
<thead>
<tr>
<th>Deterministic</th>
<th>Chance</th>
</tr>
</thead>
<tbody>
<tr>
<td>perfect information</td>
<td>chess, checkers, go, othello</td>
</tr>
<tr>
<td>imperfect information</td>
<td>stratego, 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 \textit{policy}: \( S \rightarrow A \)

Zero-Sum Games

- 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

Tic-tac-toe Game Tree

- 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
      = \textit{best achievable utility against best play}

Previously: Single-Agent Trees

- Slide from Dan Klein & Pieter Abbeel - ai.berkeley.edu
Previously: Value of a State

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

Non-Terminal States:

Non-terminal states: $V(s) = \max_{A(s)} V'(s')$

Terminal States:

Terminal states: $V(s) = \text{base}$

Adversarial Game Trees

Minimax Values

States Under Agent’s Control:

Minimax values: computed recursively

States Under Opponent’s Control:

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

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

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

Concrete Minimax Example
Minimax Example

\[
\begin{array}{c}
\text{max} \\
\text{min}
\end{array}
\]

Minimax Properties

- Optimal?
  - Yes, against perfect player. Otherwise?
- Time complexity?
  - \(O(b^m))\)
- Space complexity?
  - \(O(m)\)
- For chess, \(b \sim 35, m \sim 100\)

Do We Need to Evaluate Every Node?

\(\alpha-\beta\) Pruning Example

\(\alpha-\beta\) Pruning

- General configuration
  - \(\alpha\) 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

Progress of search...

Alpha-Beta Implementation

\[
\begin{align*}
def \text{max-value}(state, \alpha, \beta): & \\
\text{initialize } v = -\infty & \\
\text{for each successor of state: } & \\
v = \text{max}(v, \text{value}(successor, \alpha, \beta)) & \\
\text{if } v \geq \beta \text{ return } v & \\
\alpha = \text{max}(\alpha, v) & \\
\text{return } v &
\end{align*}
\]

\[
\begin{align*}
def \text{min-value}(state, \alpha, \beta): & \\
\text{initialize } v = +\infty & \\
\text{for each successor of state: } & \\
v = \text{min}(v, \text{value}(successor, \alpha, \beta)) & \\
\text{if } v \leq \alpha \text{ return } v & \\
\beta = \text{min}(\beta, v) & \\
\text{return } v &
\end{align*}
\]
Alpha-Beta Pruning Example

At max node:
- Prune if \( v \geq \beta \);
- Update \( \alpha \leq v \).

At min node:
- Prune if \( \alpha \leq v \);
- Update \( \beta \geq v \).

\( \alpha \) is MAX’s best alternative here or above
\( \beta \) is MIN’s best alternative here or above

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 bounds
- Good child ordering improves effectiveness of pruning.
  - With “perfect ordering”:
    - Time complexity drops to \( O(b^{m/2}) \)
    - Doubles solvable depth!
    - Full search of, 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 100 seconds, can explore 10K nodes / sec.
  - So can check 1M nodes per move.
  - \( \alpha-\beta \) reaches about depth 8 -- decent chess program.
- Guarantee of optimal play is gone.
- More plies makes a BIG difference.
- Use iterative deepening for an anytime algorithm.

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.
- An important example of the tradeoff between complexity of features and complexity of computation.

Demo: depth limited (UD4)
Heuristic Evaluation Function

- Function which scores non-terminals
- Ideal function: returns the utility of the position
- In practice: typically weighted linear sum of features:
  \[ \text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) \]

Evaluation for Pacman

What features would be good for Pacman?

\[ \text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) \]

Which algorithm?

- \( \alpha\beta \), \( \delta\varepsilon\eta \), \( \sigma\mu\pi\\nu\lambda \)

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

Iterative Deepening

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" failed, do a DFS which only searches paths of length 2 or less.
3. If "2" failed, do a DFS which only searches paths of length 3 or less.
   …and so on.

Why do we want to do this for multiplayer games?
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?

<table>
<thead>
<tr>
<th>Expectimax</th>
<th>Minimax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>max</strong></td>
<td><strong>average</strong></td>
</tr>
</tbody>
</table>

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 chose 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=none) = 0.25, P(T=light) = 0.55, P(T=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=heavy) = 0.20, P(T=heavy | 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(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!
Reminder: Expectations

- We can define function f(X) of a random variable X
- The expected value of a function is its average value, weighted by the probability distribution over inputs
- Example: How long to get to the airport?
  - Length of driving time as a function of traffic:
    - L(none) = 20, L(light) = 30, L(heavy) = 60
  - What is my expected driving time?
    - Notation: \( E_L(T) \)
    - Remember, P(T) = {none: 0.25, light: 0.5, heavy: 0.25}
    - \( E_L(T) = L(none) \cdot P(none) + L(light) \cdot P(light) + L(heavy) \cdot P(heavy) \)
    - \( E_L(T) = (20 \cdot 0.25) + (30 \cdot 0.5) + (60 \cdot 0.25) = 35 \)

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...

Stochastic Two-Player

- E.g. backgammon
- Expectiminimax (!)
  - Environment is an extra player that moves after each agent
  - Chance nodes take expectations
  - if state is a MAX node then return the highest \( E_{\text{max}} \) of Successors[state]
  - if state is a MIN node then return the lowest \( E_{\text{max}} \) of Successors[state]
  - if state is a chance node then return average of \( E_{\text{max}} \) of Successors[state]

Expectimax Search Trees

- What if we don't know what the result of an action will be? E.g.,
  - In solitaire, next card is unknown
  - In minesweeper, mine locations
  - In Pacman, the ghosts act randomly
- Can do expectimax search
  - Chance nodes, like min nodes, except the outcome is uncertain
  - Calculate expected utilities
  - Max nodes as in minimax search
  - Chance nodes take average (expectation) of value of children
- Later, we’ll learn how to formalize the underlying problem as a Markov Decision Process

Which Algorithm?

Minimax: no point in trying

3 ply look ahead, ghosts move randomly
Which Algorithm?

Expectimax: wins some of the time

3 ply look ahead, ghosts move randomly

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)
- Model could be sophisticated and require a great deal of computation
- We have a node for every outcome out of our control: opponent or environment
- The model might say that adversarial actions are likely!
- For now, assume for any state we magically have a distribution to assign probabilities to opponent actions / environment outcomes

Expectimax Pseudocode

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

- Notice 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-ply minimax and taking the result 80% of the time, otherwise moving randomly?

Expectimax Pruning?

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

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 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
  - Expectiminimax
    - Environment is an extra player that moves after each agent
    - Chance nodes take expectations, otherwise like minimax

Stochastic Two-Player

- Dice rolls increase b: 21 possible rolls with 2 dice
  - Backgammon = 20 legal moves
  - Depth 4 = 20 x (21 x 20)^2 x 2 x 10^6
  - As depth increases, probability of reaching a given node shrinks
  - So value of lookahead is diminished
  - But pruning is less possible...
- TDGammon uses depth-2 search + very good eval function + reinforcement learning: world-champion level play

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...