CS 188: Artificial Intelligence

Adversarial Search

Curran, Peter Abbeel
University of California, Berkeley

Game Playing State-of-the-Art

- **Checkers**: 1992: First computer player beats first computer champion. 1997: First computer solves complete 8-piece endgame. 2007: Checkers solved!
- **Chess**: 1997: Deep Blue defeats human champion. 1999: Deep Blue wins a single game match. 2007: Deeper Blue: 1000,000,000,000,000,000 evaluations per second. Sophisticated evaluation and unorthodox methods for extending search to larger trees. Current programs are even better, if less historic.
- **Go**: Human champions are now starting to be challenged by machines, though the best humans still beat the best machines. 1986: 3×3. 1995: 5×5. 2007: 5×5. Latest advances use Monte Carlo sampling methods.
- **Pacman**: Current programs are even better, if less historic.

Types of 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

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: SxA → S
  - Terminal Test: S → {0,1}
  - Terminal Utilities: S → R
- Solution for a player is a **policy**: S → A
Zero-Sum Games
- Zero-Sum Games:
  - Agents have opposite utilities (values on outcomes)
  - Let’s 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

Adversarial Search

Single-Agent Trees

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

Adversarial Game Trees

Minimax Values
- States Under Agent’s Control:
  \[ V(s) = \max_{\pi_{\text{Agent}}} \min_{\pi_{\text{Opponent}}} V'(s') \]
- States Under Opponent’s Control:
  \[ V'(s') = \min_{\pi_{\text{Agent}}} \max_{\pi_{\text{Opponent}}} V(s) \]
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

Algorithm:

**Minimax Implementation**

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

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

**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)
```

**Minimax Example**

```
1 | 2 | 3
2 | 3 | 4
3 | 4 | 5
```

**Minimax Efficiency**

- How efficient is minimax?
  - Just like (exhaustive) DFS
  - Time: \(O(b^m)\)
  - Space: \(O(bm)\)

- Example: For chess, \(b \approx 35, m \approx 100\)
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree?
Minimax Properties

Optimal against a perfect player. Otherwise?

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
- α-β 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

Evaluation Functions

- Evaluation functions score non-terminals in depth-limited search
- Ideal function: returns the actual minimax value of the position
- In practice: typically weighted linear sum of features:
  \[ \text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) \]
- e.g. \[ f_1(s) = \text{num white queens} - \text{num black queens} \]
- Examples: Stuart Russell
Evaluation for Pacman

Why Pacman Starves

- A danger of replanning agents!
  - He knows his score will go up by eating the dot now (west, east)
  - He knows his score will go up just as much by eating the dot later (east, west)
  - There are no point-scoring opportunities after eating the dot (within the horizon, two here)
  - Therefore, waiting seems just as good as eating; he may go east, then back west in the next round of replanning!

Game Tree Pruning

Minimax Example

Minimax Pruning

Alpha-Beta Pruning

- General configuration (MIN version)
  - We're computing the MIN/VALUE at some node n
  - We're looping over n's children
  - n's estimate of the children's min is dropping
  - Who cares about n's value? MAX
  - Let a be the best value that MAX can get at any choice point along the current path from the root
  - If n becomes worse than a, MAX will avoid it, so we can stop considering n's other children (it's already bad enough that it won't be played)
- MAX version is symmetric
**Alpha-Beta Implementation**

```python
def max_value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
    if v ≥ β return v
    α = max(α, v)
    return v

def min_value(state, α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
    if v ≤ α return v
    β = min(β, v)
    return v
```

**Alpha-Beta Pruning Properties**

- This pruning has **no effect** on minimax value computed for the root!
- Values of intermediate nodes might be wrong
  - Important: children of the root may have the wrong value
  - So the naive version won’t let you do action selection
- 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...
- This is a simple example of **metareasoning** (computing about what to compute)

**Uncertain Outcomes**

- 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
  - 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

**Worst-Case vs. Average Case**

- Idea: Uncertain outcomes controlled by chance, not an adversary!
**Expectimax Pseudocode**

- **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 EXP: return exp-value(state)

- **def exp-value(state):**
  - initialize v = 0
  - for each successor of state:
    - p = probability(successor)
    - v += p * value(successor)
  - return v

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

**Expectimax Example**

\[ v = \frac{1}{2} \times 8 + \frac{1}{3} \times 24 + \frac{1}{6} \times -12 = 10 \]

**Expectimax Pruning?**

**Depth-Limited Expectimax**

**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 ∈ {none, light, heavy}
- Distribution: P(T=none) = 0.25, P(T=light) = 0.50, P(T=heavy) = 0.25

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:

Example: Traffic on freeway

Probabilities over all possible outcomes sum to one

Dangerous Pessimism

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?

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 (not a probability distribution)

Model could be sophisticated and require a great deal of computation

We assign a chance node for any outcome out of our control

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.

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, which has the nice property that it all collapses into one game tree.
**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.

**Other Game Types**

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

<table>
<thead>
<tr>
<th>Expectimax Pacman</th>
<th>Random Ghost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Win 5/5</td>
<td>Win 5/5</td>
</tr>
<tr>
<td>Avg. Score: -303</td>
<td>Avg. Score: 503</td>
</tr>
</tbody>
</table>

Results from playing 5 games.

**Mixed Layer Types**

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

**Example: Backgammon**

- Dice rolls increase in 21 possible rolls with 2 dice
- Backgammon = 20 legal moves
- Depth 2 = 20 x (21 x 20) = 1.2 x 10^7
- 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 tricky...
- 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:
  - Terminal states have utility tuples
  - Node values are also utility tuples
  - Each player maximizes its own component
  - Can give rise to cooperation and competition dynamically...

**Utilities**
Maximum Expected Utility

- Why should we average utilities? Why not maxmin?
- Principle of maximum expected utility
  - A rational agent should choose the action that maximizes its expected utility, given its knowledge
- Questions:
  - Where do utilities come from?
  - How do we know such utilities exist?
  - How do we know that averaging even makes sense?
  - What if our behavior (preferences) can’t be described by utilities?

What Utilities to Use?

- For worst-case minmax reasoning, terminal 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 average-case expectimax reasoning, we need magnitudes to be meaningful

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 “rational” preferences can be summarized by a utility function
- We hard-wire utilities and let behaviors emerge
  - Why don’t we let agents pick utilities?
  - Why don’t we prescribe behaviors?

Utilities: Uncertain Outcomes

- An agent must have preferences among:
  - Prizes: A, B, etc.
  - Lotteries: situations with uncertain prizes
  - Notations:
    - Preference: \( A > B \)
    - Indifference: \( A \sim B \)

Preferences

- An agent must have preferences among:
  - Prizes: A, B, etc.
  - Lotteries: situations with uncertain prizes
  - Notations:
    - Preference: \( A > B \)
    - Indifference: \( A \sim B \)
Rational Preferences

- We want some constraints on preferences before we call them rational, such as:
  - Axiom of Transitivity: \((A > B) \land (B > C) \implies (A > C)\)

- For example: an agent with intransitive preferences can be induced to give away all of its money:
  - If \(B > C\), then an agent with \(C\) would pay (say) 1 cent to get \(B\)
  - If \(A > B\), then an agent with \(B\) would pay (say) 1 cent to get \(A\)
  - If \(C > A\), then an agent with \(A\) would pay (say) 1 cent to get \(C\)

MEU Principle

- Theorem (Ramsey, 1931; von Neumann & Morgenstern, 1944):
  - Given any preferences satisfying these constraints, there exists a real-valued function \(U\) such that:
    \[
    U(A) \geq U(B) \iff A \succeq B
    \]
    \[
    U(p_1S_1 + \ldots + p_nS_n) = \sum p_i U(S_i)
    \]
  - i.e., values assigned by \(U\) preserve preferences of both prizes and lotteries

- Maximum expected utility (MEU) principle:
  - Choose the action that maximizes expected utility
  - Note: an agent can be entirely rational (consistent with MEU) without ever representing or displaying it
  - E.g., a lookup table for perfect tic-tac-toe, a reflex vacuum cleaner

Utility Scales

- Normalized utilities: \(u_e = 1.0, u_0 = 0.0\)
- Micromorts: one-millionth chance of death, useful for paying to reduce product risks, etc.
- QALYs: quality-adjusted life years, useful for medical decisions involving substantial risk
- Note: behavior is invariant under positive linear transformation
  \[
  U'(x) = k_1 U(x) + k_2 \quad \text{where } k_1 > 0
  \]
- With deterministic prizes (no lottery choices), only ordinal utility can be determined, i.e., total order on prizes

Human Utilities

- Utilities map states to real numbers. Which numbers?
- Standard approach to assessment (elicitation) of human utilities:
  - Compare a prize \(A\) to a standard lottery \(L\), between:
    - “Best possible prize” \(u_e\) with probability 1
    - “Worst possible catastrophe” \(u_0\) with probability 1
  - Adjust lottery probability \(p\) until indifference: \(A \sim L\)
  - Resulting \(p\) is a utility in [0,1]

  Pay $30

  0.999999
  0.000001

  No change
  Instant death
Money

- Money does not behave as a utility function, but we can talk about the utility of having money (or being in debt).
- Given a lottery \( L = [p, $X; (1-p), $Y] \)
  - The expected monetary value (EMV) is \( pX + (1-p)Y \)
  - Typically, \( U(L) \leq U(EMV(L)) \)
  - In this sense, people are risk-averse.
  - When deep in debt, people are risk-prone.

Example: Insurance

- Consider the lottery \([0.5, $1000; 0.5, $0]\)
- What is its expected monetary value? \( ($500) \)
- What is its certainty equivalent? \( $400 \) for most people
- Difference of $100 is the insurance premium.
- There’s an insurance industry because people will pay to reduce their risk.
- If everyone were risk-neutral, no insurance needed!
- It’s win-win: you’d rather have the $400 and the insurance company would rather have the lottery (their utility curve is flat and they have many lotteries).

Example: Human Rationality?

- Famous example of Allais (1953)
  - A: \([0.8, $4k; 0.2, $0]\)
  - B: \([1.0, $3k; 0.0, $0]\)
  - C: \([0.2, $4k; 0.8, $0]\)
  - D: \([0.25, $3k; 0.75, $0]\)
  - Most people prefer \( B > A, C > D \)
  - But if \( U($0) = 0 \), then
    - \( B > A \implies U($4k) > 0.8 U($4k) \)
    - \( C > D \implies 0.8 U($4k) > U($3k) \)