CIS 471/571 (Winter 2020): Introduction to Artificial Intelligence

Lecture 6: Adversarial Search

Thanh H. Nguyen

Source: http://ai.berkeley.edu/home.html
Announcements

- Project 2:
  - Deadline: Feb 02, 2020

- Homework 2:
  - Deadline: Feb 03, 2020
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
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

- 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

```
  2  0  ...
2  6  ...
  4  6
  8
```
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:
\[ V(s) = \text{known} \]
Adversarial Game Trees
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 values: computed recursively

Terminal values: part of the game
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
Minimax Properties

Optimal against a perfect player. Otherwise?
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?
Resource Limits
Game Tree Pruning
Minimax Example
Minimax Pruning
Alpha-Beta Pruning

- **Alpha** $\alpha$: value of the best choice so far for MAX (lower bound of Max utility)
- **Beta** $\beta$: value of the best choice so far for MIN (upper bound of Min utility)

Expanding at MAX node $n$: update $\alpha$
- If a child of $n$ has value greater than $\beta$, stop expanding the MAX node $n$
- Reason: MIN parent of $n$ would not choose the action which leads to $n$

At MIN node $n$: update $\beta$
- If a child of $n$ has value less than $\alpha$, stop expanding the MIN node $n$
- Reason: MAX parent of $n$ would not choose the action which leads to $n$
def min-value(state, α, β):
    initialize $v = +\infty$
    for each successor of state:
        $v = \min(v, \text{value(successor, α, β)})$
    if $v \leq α$ return $v$
    $β = \min(β, v)$
    return $v$

def max-value(state, α, β):
    initialize $v = -\infty$
    for each successor of state:
        $v = \max(v, \text{value(successor, α, β)})$
    if $v \geq β$ return $v$
    $α = \max(α, v)$
    return $v$

def value(state, α, β):
    if the state is a terminal state: return the state’s utility
    if the next agent is MAX: return max-value(state, α, β)
    if the next agent is MIN: return min-value(state, α, β)
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 most naïve version won’t let you do action selection

- Good child ordering improves effectiveness of pruning
Alpha-Beta Quiz

$[\alpha, \beta] = [\infty, +\infty]$
Alpha-Beta Quiz

[max, min] = [−∞, +∞]

[max]

[min, max] = [−∞, +∞]

10  8  4  50
Alpha-Beta Quiz

\[
[\alpha, \beta] = [-\infty, +\infty]
\]

\[
[\alpha, \beta] = [-\infty, 10]
\]
Alpha-Beta Quiz

\[ [\alpha, \beta] = [-\infty, +\infty] \]

\[ [\alpha, \beta] = [-\infty, 8] \]
**Alpha-Beta Quiz**

\[ [\alpha, \beta] = [-\infty, +\infty] \]

\[ [\alpha, \beta] = [-\infty, 8] \]
Alpha-Beta Quiz

\[ [\alpha, \beta] = [8, +\infty] \]

\[ [\alpha, \beta] = [-\infty, 8] \]
Alpha-Beta Quiz

\[
[\alpha, \beta] = [8, +\infty]
\]

\[
[\alpha, \beta] = [-\infty, 8]
\]

\[
[\alpha, \beta] = [8, +\infty]
\]

[Image of a game tree with nodes labeled a, b, c, d, e, f and values 10, 8, 4, 50]
Alpha-Beta Quiz

The diagram represents an Alpha-Beta search algorithm with the following min-max values:

**Max Node Values:**
- \([\alpha, \beta] = [8, +\infty]\)

**Min Node Values:**
- \([\alpha, \beta] = [-\infty, 8]\)
- \([\alpha, \beta] = [8, +\infty]\)

The diagram shows the decision-making process with nodes labeled with values and connected by arrows indicating the flow of the search.
**Alpha-Beta Quiz**

\[ [\alpha, \beta] = [8, +\infty] \]

\[ [\alpha, \beta] = [-\infty, 8] \]

\[ [\alpha, \beta] = [8, +\infty] \]
Alpha-Beta Quiz 2
Alpha-Beta Quiz 2

\[ [\alpha, \beta] = [-\infty, +\infty] \]
Alpha–Beta Quiz 2

\[ [\alpha, \beta] = [-\infty, +\infty] \]
Alpha–Beta Quiz 2

\[ [\alpha, \beta] = [\infty, +\infty] \]

\[ [\alpha, \beta] = [\infty, +\infty] \]

\[ [\alpha, \beta] = [10, +\infty] \]

\[ 10 \quad 6 \quad 100 \quad 8 \quad 1 \quad 2 \quad 20 \quad 4 \]
Alpha-Beta Quiz 2

\[ [\alpha, \beta] = [-\infty, +\infty] \]

\[ [\alpha, \beta] = [10, +\infty] \]

\[ [\alpha, \beta] = [-\infty, +\infty] \]
Alpha-Beta Quiz 2

\[
[\alpha, \beta] = [-\infty, +\infty]
\]

\[
[\alpha, \beta] = [10, +\infty]
\]

\[
[\alpha, \beta] = [-\infty, +\infty]
\]
Alpha-Beta Quiz 2

\[ [\alpha, \beta] = [-\infty, +\infty] \]

\[ [\alpha, \beta] = [-\infty, 10] \]

\[ [\alpha, \beta] = [10, +\infty] \]

10 6 100 8 1 2 20 4
Alpha-Beta Quiz 2

\[
\begin{align*}
[\alpha, \beta] &= [-\infty, +\infty] \\
[\alpha, \beta] &= [-\infty, 10] \\
[\alpha, \beta] &= [10, +\infty] \\
[\alpha, \beta] &= [-\infty, 10] \\
[\alpha, \beta] &= [10, +\infty] \\
\end{align*}
\]
\[
[\alpha, \beta] = [\infty, +\infty]
\]
\[
[\alpha, \beta] = [-\infty, 10]
\]
\[
[\alpha, \beta] = [10, +\infty]
\]
\[
[\alpha, \beta] = [-\infty, 10]
\]
Alpha-Beta Quiz 2

[Image of a decision tree with labels and values]

- $[\alpha, \beta] = [-\infty, +\infty]$
- $[\alpha, \beta] = (-\infty, 10]$
Alpha–Beta Quiz 2

\[
[\alpha, \beta] = [-\infty, +\infty]
\]

\[
[\alpha, \beta] = [-\infty, 10]
\]

\[
[\alpha, \beta] = [10, +\infty]
\]

\[
[\alpha, \beta] = [-\infty, 10]
\]

\[
[\alpha, \beta] = [10, +\infty]
\]

\[
[\alpha, \beta] = [-\infty, 10]
\]
\[
[[\alpha, \beta]] = [10, +\infty]
\]

\[
[[\alpha, \beta]] = [-\infty, 10]
\]

\[
[[\alpha, \beta]] = [10, +\infty]
\]

\[
[[\alpha, \beta]] = [10, +\infty]
\]

\[
[[\alpha, \beta]] = [-\infty, 10]
\]

\[
[[\alpha, \beta]] = [10, +\infty]
\]
Alpha-Beta Quiz 2

\[ [\alpha, \beta] = [10, +\infty] \]

\[ [\alpha, \beta] = [10, +\infty] \]

\[ [\alpha, \beta] = [-\infty, 10] \]

\[ [\alpha, \beta] = [-\infty, 10] \]

\[ [\alpha, \beta] = [10, +\infty] \]

\[ [\alpha, \beta] = [10, +\infty] \]
\[ [\alpha, \beta] = [-\infty, 10] \]

\[ [\alpha, \beta] = [10, +\infty] \]

\[ [\alpha, \beta] = [-\infty, 10] \]

\[ [\alpha, \beta] = [10, +\infty] \]

\[ [\alpha, \beta] = [-\infty, 10] \]

\[ [\alpha, \beta] = [10, +\infty] \]

\[ [\alpha, \beta] = [10, +\infty] \]
**Alpha-Beta Quiz 2**

![Alpha-Beta Tree Diagram]

Maximal values:
- $[\alpha, \beta] = [10, +\infty]$ on the right
- $[\alpha, \beta] = [10, +\infty]$ on the bottom

Minimal values:
- $[\alpha, \beta] = [-\infty, 10]$ on the left
- $[\alpha, \beta] = [-\infty, 10]$ on the bottom

The diagram shows decision points with minimal and maximal values, leading to leaf nodes with values 10, 6, 100, 8, 1, 2, 20, and 4.
Alpha-Beta Quiz 2

$[\alpha, \beta] = [10, +\infty]$
The given image contains a diagram representing a decision-making process related to the Alpha-Beta Quiz 2. The diagram illustrates a game tree with nodes representing decisions and outcomes. The values at the nodes are denoted as follows:

- The node labeled "10" at the top has a min-max value of $[\alpha, \beta] = [-\infty, 10]$.
- The node labeled "2" has a min-max value of $[\alpha, \beta] = [10, +\infty]$.
- The node labeled "10" at the bottom has a min-max value of $[\alpha, \beta] = [10, +\infty]$.
- The node labeled "6" has a min-max value of $[\alpha, \beta] = [-\infty, 10]$.
- The node labeled "100" has a min-max value of $[\alpha, \beta] = [-\infty, 10]$.
- The node labeled "8" has a min-max value of $[\alpha, \beta] = [10, +\infty]$.
- The node labeled "1" has a min-max value of $[\alpha, \beta] = [10, +\infty]$.
- The node labeled "2" has a min-max value of $[\alpha, \beta] = [10, +\infty]$.
- The node labeled "20" has a min-max value of $[\alpha, \beta] = [10, +\infty]$.
- The node labeled "4" has a min-max value of $[\alpha, \beta] = [10, +\infty]$.

The diagram shows the decision-making process with optimal strategies for both max and min players, with some branches representing suboptimal decisions.
Resource Limits
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
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!
Evaluation Functions
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. \( f_1(s) = (\text{num white queens} - \text{num black queens}) \), etc.
Evaluation for Pacman
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 (L6D4, L6D5)]
Synergies between Evaluation Function and Alpha-Beta?

- **Alpha-Beta**: amount of pruning depends on expansion ordering
  - Evaluation function can provide guidance to expand most promising nodes first (which later makes it more likely there is already a good alternative on the path to the root)
    - (somewhat similar to role of A* heuristic, CSPs filtering)

- **Alpha-Beta**: (similar for roles of min-max swapped)
  - Value at a min-node will only keep going down
  - Once value of min-node lower than better option for max along path to root, can prune
  - Hence: IF evaluation function provides upper-bound on value at min-node, and upper-bound already lower than better option for max along path to root THEN can prune