CIS 471/571 (Winter 2019): Introduction to Artificial Intelligence

Lecture 6: Adversarial Search

Thanh H. Nguyen

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

- Project 1: Search (Reminder)
  - Deadline: Oct 18th, 2019

- Homework 2:
  - Deadline: Oct 30th, 2019
Today

- Finish CSPs

- Game tree: minimax
Tree Decomposition*

- Idea: create a tree-structured graph of mega-variables
- Each mega-variable encodes part of the original CSP
- Sub-problems overlap to ensure consistent solutions

\[(M_1,M_2) \in \{(WA=g,SA=g,NT=g), (NT=g,SA=g,Q=g)\}, \ldots\]
Iterative Improvement
Iterative Algorithms for CSPs

- Local search methods typically work with “complete” states, i.e., all variables assigned

- To apply to CSPs:
  - Take an assignment with unsatisfied constraints
  - Operators *reassign* variable values
  - No fringe! Live on the edge.

- Algorithm: While not solved,
  - Variable selection: randomly select any conflicted variable
  - Value selection: min-conflicts heuristic:
    - Choose a value that violates the fewest constraints
    - I.e., hill climb with \( h(n) = \) total number of violated constraints
Example: 4-Queens

- States: 4 queens in 4 columns ($4^4 = 256$ states)
- Operators: move queen in column
- Goal test: no attacks
- Evaluation: $c(n) =$ number of attacks
Performance of Min-Conflicts

- Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., n = 10,000,000)!

- The same appears to be true for any randomly-generated CSP except in a narrow range of the ratio

\[ R = \frac{\text{number of constraints}}{\text{number of variables}} \]
Summary: CSPs

- CSPs are a special kind of search problem:
  - States are partial assignments
  - Goal test defined by constraints

- Basic solution: backtracking search

- Speed-ups:
  - Ordering
  - Filtering
  - Structure

- Iterative min-conflicts is often effective in practice
Local Search
Local Search

- Tree search keeps unexplored alternatives on the fringe (ensures completeness)

- Local search: improve a single option until you can’t make it better (no fringe!)

- New successor function: local changes

- Generally much faster and more memory efficient (but incomplete and suboptimal)
Hill Climbing

- Simple, general idea:
  - Start wherever
  - Repeat: move to the best neighboring state
  - If no neighbors better than current, quit

- What’s bad about this approach?
  - Complete?
  - Optimal?

- What’s good about it?
Hill Climbing Diagram

- Objective function
- Global maximum
- Shoulder
- Local maximum
- "Flat" local maximum
- Current state
- State space
Hill Climbing Quiz

Starting from X, where do you end up?

Starting from Y, where do you end up?

Starting from Z, where do you end up?
Simulated Annealing

- Idea: Escape local maxima by allowing downhill moves
  - But make them rarer as time goes on

```plaintext
function SIMULATED-ANNEALING(problem, schedule) returns a solution state

inputs: problem, a problem
         schedule, a mapping from time to “temperature”

local variables: current, a node
                 next, a node
                 T, a “temperature” controlling prob. of downward steps

current ← MAKE-NODE(Initial-State[problem])
for t ← 1 to ∞ do
    T ← schedule[t]
    if T = 0 then return current
    next ← a randomly selected successor of current
    ΔE ← VALUE[next] − VALUE[current]
    if ΔE > 0 then current ← next
    else current ← next only with probability $e^{ΔE/T}$
```
Adversarial Games
Many different kinds of games!

Aaxes:
- 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: $S \times A \to S$
  - Terminal Test: $S \to \{t,f\}$
  - Terminal Utilities: $S \times P \to R$

- Solution for a player is a policy: $S \to A$
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
Adversarial Search
Single-Agent Trees
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

-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

MAX (X)

MIN (O)

MAX (X)

MIN (O)

TERMINAL

Utility

-1  0  +1
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
Minimax Implementation

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 = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v

def min-value(state):
    initialize v = +∞
    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

![Minimax Pruning Diagram]

**MAX**

**MIN**

- 3
- 12
- 8
- 2

- 14
- 5
- 2
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 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)

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 most naïve version won’t let you do action selection

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

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

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

[Diagram showing a game tree with nodes labeled 'max' and 'min', and values 10, 8, 4, and 50 at the leaves.]
Alpha-Beta Quiz

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

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

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

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

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

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

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

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

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

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

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

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

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

\[
\begin{align*}
[\alpha, \beta] &= [8, +\infty] \\
\min & \quad [\alpha, \beta] = [-\infty, 8] \\
\max & \quad [\alpha, \beta] = [8, +\infty]
\end{align*}
\]
Alpha–Beta Quiz 2

![Alpha-Beta Tree Diagram]

- **Max** and **Min** nodes in the tree.
- **Numbers** at the leaf nodes: 10, 6, 100, 8, 1, 2, 20, 4.
$$[\alpha, \beta] = [-\infty, +\infty]$$
Alpha-Beta Quiz 2

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

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

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

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

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

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

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

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

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

10

6

100

8

1

2

20

4
Alpha-Beta Quiz 2

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

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

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

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

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

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

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

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

\[
[\alpha, \beta] = [100, +\infty]
\]

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

\[
[\alpha, \beta] = [20, +\infty]
\]

\[
[\alpha, \beta] = [4, +\infty]
\]

\[
[\alpha, \beta] = [2, +\infty]
\]

\[
[\alpha, \beta] = [1, +\infty]
\]

\[
[\alpha, \beta] = [6, +\infty]
\]

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

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

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

\[\begin{align*}
\alpha, \beta & = [10, +\infty] \\
[\alpha, \beta] & = [-\infty, 10] \\
[\alpha, \beta] & = [-\infty, +\infty]
\end{align*}\]
\[ [\alpha, \beta] = \left[ -\infty, +\infty \right] \]

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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