CSC242: Intro to AI

Lecture 6
TA Help Sessions (v2)

- Monday & Tuesday: 17:00–18:00, Hylan 301
- Doodle poll signup before 16:00
- Link on BB: http://www.doodle.com/xgxcbxn4knks86sx
Local Search

- Evaluates and modifies a small number of current states
- Does not record history of search (paths, explored set, etc.)

**Good:** Very little (constant) memory

**Bad:** May not explore all alternatives

=> Incomplete, not optimal
Local Search
Hill-Climbing (Greedy Local Search)

- Move through state space in the direction of increasing value ("uphill")
current state

objective function

global maximum

shoulder

local maximum

“flat” local maximum

state space
Random Restarts

• If at first you don’t succeed, try again!
• From a random initial state...
Stochastic Local Search

• Or by randomizing the generation of successors
Simulated Annealing

- Follow landscape down towards global minimum of state cost function
- Occasionally allow an upward move ("shake") to get out of local minima
- Don’t shake so hard that you bounce out of global minimum
Simulated Annealing

Complete?  No.
Optimal?  No.

“But if the schedule lowers T slowly enough, simulated annealing will find a global minimum with probability approaching one.”
Local Beam Search

- During hill-climbing:
  - Keep track of $k$ states rather than just one
  - At each step, generate all successors of all $k$ states ($k^b$ of them)
  - Keep the most promising $k$ of them
Parallel Local Search
Parallel Local Search
Local Beam Search
Local Beam Search
“And this report just in. ... Apparently, the grass is greener on the other side.”
Asexual Reproduction
Mitosis
Sexual Reproduction
Meiosis
Genetic Algorithms

- Start with $k$ random states
- Select pairs of states and have them “mate” to produce “offspring”
- Most fit (highest-scoring) individuals reproduce more often
Genetic Algorithms

• States encoded as “chromosomes” (linear sequences, a.k.a. strings)

• During mating:
  • Crossover: swap chunks of code
  • Mutation: randomly change bits of code
Genetic Algorithms

- States encoded as “chromosomes” (linear sequences, a.k.a. strings)
- During mating:
  - Crossover: swap chunks of code
  - Mutation: randomly change bits of code
GA Summary

• A version of stochastic local beam search with a special way to generate successor states (motivated by a naive biology analogy)

• “Much work remains to be done to identify the conditions under which genetic algorithms perform well.”
- Evaluates and modifies a small number of current states
- Does not record history of search

Hill-climbing

**Good:** Very little (constant) memory

**Bad:** May not explore all alternatives

=> Incomplete

Local Beam Search

Simulated Annealing

Genetic Algorithms
Quiz Thursday on Local Search
Adversarial Search
Adversarial Search
Battle of The Bulge
16-25 Dec 1944
Multi-Agent Environments

• Unpredictability of other agents => contingencies (strategies)

• Agents goals in conflict => competition
Games
Games
Games

• “Require the ability to make some decision even when calculating the optimal decision is infeasible”

• “Penalize inefficiency severely”
Games ≠ Toy Problems

$9! = 362880$

$35^{100} = 10^{154}$

$2 \times 10^{170}$
Abstract Games
Abstract Games

- Deterministic (no chance)
- Nondeterministic (dice, cards, etc.)
Abstract Games

- Perfect information (fully observable)
- Imperfect information (partially observable)
Abstract Games

- Zero-sum (total payoff the same in any game)
- What’s good for me is bad for you and vice-versa
- Arbitrary utility function
## Types of Games

<table>
<thead>
<tr>
<th>Deterministic (no chance)</th>
<th>Nondeterministic (dice, cards, etc.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perfect information (fully observable)</td>
<td>Imperfect information (partially observable)</td>
</tr>
<tr>
<td>Zero-sum (total payoff the same in any game)</td>
<td>Arbitrary utility functions</td>
</tr>
</tbody>
</table>
Outline

• In deterministic, perfect information, zero-sum games:
  • How to find optimal moves in deterministic games
  • How to find good moves when time is limited

• Next time: nondeterministic

• Later: imperfect information, utilities
Next Player: O
Deciding what to do in a game requires thinking about what the opponent will do, and having a strategy that takes that into account. Not simply a sequence of actions, but a contingency plan that specifies what to do depending on what state one finds oneself in (AIMA 4.3)
Utility

-1  0  +1

Term.

0 0 0

X O X

X O X

X O X

X O X

X O X

X O X

X O X

X O X
Minimax Algorithm

$$\text{MINIMAX}(s) = \begin{cases} 
\text{UTILITY}(s) & \text{if } \text{TERMINAL-TEST}(s) \\
\max_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\
\min_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MIN} 
\end{cases}$$
Minimax Analysis

Time Complexity: $O(b^m)$

Space Complexity: $O(bm)$
Minimax Summary

• Computes the optimal move assuming opponent also plays optimally (i.e., worst-case outcome)

• Explores game tree depth-first all the way to terminal states (end of game)

• Backs up utility values through alternating MIN and MAX (what’s best for me is worst for you, and vice-versa)
Minimax Code

AIMA 5.2.1 and Figure 5.3 (page 166)

Code template on course website and linked in BB announcement
A Problem for Minimax

• You can’t search all the way to the terminal nodes

• You can’t evaluate a node (state) unless you’re at a terminal node

• Utility function is defined on terminal states
Imperfect Real-Time Decisions
$h(n)$
**Minimax Algorithm**

\[
\text{Minimax}(s) =
\begin{cases} 
\text{Utility}(s) & \text{if Terminal-Test}(s) \\
\max_{a \in \text{Actions}(s)} \text{Minimax}(\text{Result}(s, a)) & \text{if Player}(s) = \max \\
\min_{a \in \text{Actions}(s)} \text{Minimax}(\text{Result}(s, a)) & \text{if Player}(s) = \min
\end{cases}
\]
Heuristic Minimax

\[ H_{-\text{MINIMAX}}(s) = \begin{cases} h(s) & \text{if \ CUTOFF-TEST}(s) \\ \max\limits_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if \ PLAYER}(s) = \text{MAX} \\ \min\limits_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if \ PLAYER}(s) = \text{MIN} \end{cases} \]
H-Minimax: Cutoff

- When to cutoff search?
  - Time
  - Depth
  - “Quiescence”
- Can adjust dynamically
- AIMA 5.4.2
H-Minimax: Heuristic

- How to evaluate non-terminal state?
- AIMA 5.4.1
- Chess example
  - Material value
  - Weighted sum
  - Strategic considerations
Heuristic Minimax

- Cutoff search before reaching terminal nodes (time, depth, “quiescence”)
- Use heuristic evaluation function to estimate state utility
- Backs up utility values through alternating MIN and MAX (what’s best for me is worst for you, and vice-versa)
Adversarial Search: MINIMAX

**MINIMAX**
- Searches to terminal nodes
- Uses utility function

**H-MINIMAX**
- Cuts off search before terminals
- Uses heuristic function

Backs up utility values through alternating MIN and MAX (zero-sum game)
For next time:
AIMA 5.3-5.4.2,
5.5-5.6; 5.7-5.9 fyi

Quiz on Local Search
(Ch 4 per syllabus)