Games

Andrea Danyluk
February 15, 2017

Announcements
• Programming Assignment 1: Search
  – Still in progress
  – A note about designing heuristics:
    • Add a “feature” at a time
    • Consider different weights for different features
    • Think beyond adding heuristic information together
    • Once you have a function that works well, remove elements to determine whether you really need them

Today
• Games (repeated from last time)
  – Planning/problem solving in the presence of an adversary ➔ adversarial search
  – Why games?
    • Easy to measure success or failure
    • States and rules are generally easy to specify
    • Interesting and complex
      – Space and time complexity
      – Uncertainty of adversaries’ action, rolls of dice, etc.

Go
• AlphaGo became the first program to beat a human professional Go player without handicaps on a full 19x19 board.
• In go, b > 300
• Uses Monte Carlo tree search to select moves.
• Uses knowledge learned from a combination of reinforcement and deep learning.

Backgammon
• TDGammon uses depth-2 search + very good evaluation function + reinforcement learning (Gerry Tesauro, IBM)
• World-champion level play
• 1st AI world champion in any game!

Poker
• Libratus [Sandholm and Brown, CMU] won $1.7m (in chips) from 4 professional poker players over 20 days in January 2017
• No-limit Texas Hold’em
• Hard because it’s a game of imperfect information. Can’t see the opponent’s hand.
• The “final frontier” in games...

[Adapted from CS 188 Berkeley]
Types of Games

<table>
<thead>
<tr>
<th>Perfect Information</th>
<th>Deterministic</th>
<th>Chess, Checkers, Go, Connect Four</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Chance</td>
<td>Backgammon</td>
</tr>
</tbody>
</table>

| Imperfect Information | Battleship, Guess Who? | Bridge, Poker, Scrabble |

Want algorithms for calculating a strategy (policy) that recommends a move in each state

Connect Four Demo

- With perfect play, first player can force a win by starting in the middle column.
- By starting in one of the two adjacent columns, the first player allows the second player to reach a draw.
- By starting in any of the four outer columns, the first player allows the second player to force a win.
- There exist perfect players – my demo program is not one of them.

Game Playing as a Search Problem

Note that each level in the game tree (i.e., each half move) is called a ply.

Formulating Game Playing as Search

- States S
  - Description of the current state/configuration of the game
- Players P = {1, 2, ..., n}
  - Will take turns in the games we consider
- Actions A
  - Legal actions may depend on player and state
- Transition model
  - Defines the result of an action applied to a state for a particular player
- Terminal test
  - Function on states; returns T if state is a terminal state and F otherwise
- Utility function S x P -> value
  - Also called objective function or payoff function

Games vs Search Problems

- “Unpredictable” opponent ⇒ solution is a strategy
- Time limits ⇒ unlikely to reach terminal states.
  - Must approximate
Minimax Search

- When it’s your turn, generate (ideally) the complete game tree.
- Select the move that is best for you, assuming that your opponent will, at each opportunity, select the move that is worst for you (and thus best for him/her/itself)
An Example: 2-player zero-sum game

Minimax Search revisited

- A state-space search tree
- Players alternate turns
- Each node has a minimax value: best achievable utility against a rational adversary

[Adapted from CS 188 Berkeley]
Another Example

Another Example

Another Example

Another Example

Another Example

But really done depth-first
function MINIMAX-DECISION(state) returns an action a
   return arg max a in ACTIONS(state), MIN-VALUE(RESULT(state, a))

function MIN-VALUE(state) returns a utility value v
   if TERMINAL-TEST(state) then return UTILITY(state)
   v = infinity
   for each a in ACTIONS(state) do
      v = MIN(v, MAX-VALUE(RESULT(state, a)))
   return v

function MAX-VALUE(state) returns a utility value v
   if TERMINAL-TEST(state) then return UTILITY(state)
   v = -infinity
   for each a in ACTIONS(state) do
      v = MAX(v, MIN-VALUE(RESULT(state, a)))
   return v

Minimax Reality

- Can rarely explore entire search space to terminal nodes.
  - DFS has good space complexity, but bad time complexity
- Choose a depth cutoff – i.e., a maximum ply
- Need an evaluation function
  - Returns an estimate of the expected utility of the game from a
given position
  - Must order the terminal states in the same way as the true
utility function
  - Must be efficient to compute
    * Trading off pils for heuristic computation
    * More pils makes a difference
- Consider iterative deepening
Evaluation Functions

• Ideal: returns the utility of the position
• In practice: typically weighted linear sum of features:
  • Eval(s) = w_1 f_1(s) + w_2 f_2(s) + ... + w_n f_n(s)

Exercise

• Evaluation function for Connect Four?