Adversarial Search and Game Playing

Russell and Norvig, Chapter 5

Adversarial Search and Game Playing

Games

- Games: multi-agent environment
  - What do other agents do and how do they affect our success?
  - Cooperative vs. competitive multi-agent environments.
  - Competitive multi-agent environments give rise to adversarial search a.k.a. games

- Why study games?
  - Fun!
  - They are hard
  - Easy to represent and agents restricted to small number of actions... sometimes!

Relation of Games to Search

- Search – no adversary
  - Solution is (heuristic) method for finding goal
  - Heuristics and CSP techniques can find optimal solution
  - Evaluation function: estimate of cost from start to goal through given node
  - Examples: path planning, scheduling activities

- Games – adversary
  - Solution is strategy (strategy specifies move for every possible opponent reply).
  - Time limits force approximate solutions
  - Examples: chess, checkers, Othello, backgammon

Types of Games

<table>
<thead>
<tr>
<th>Perfect information</th>
<th>Deterministic</th>
<th>Chance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chess, Go, Checkers, Othello</td>
<td>chess, go, checkers, othello</td>
<td>backgammon</td>
</tr>
<tr>
<td>Imperfect information</td>
<td>Bridge, Hearts</td>
<td>Poker, Canasta, Scrabble</td>
</tr>
</tbody>
</table>

Our focus: deterministic, turn-taking, two-player, zero-sum games of perfect information

- zero-sum game: a participant's gain (or loss) is exactly balanced by the losses (or gains) of the other participant.
- perfect information: fully observable

Game setup

- Two players: MAX and MIN
- MAX moves first and they take turns until the game is over.
- Games as search:
  - Initial state: e.g. starting board configuration
  - Player(s): which player has the move in a state
  - Action(s): set of legal moves in a state
  - Result(s, a): the states resulting from a given move.
  - Terminal-test(s): game over? (terminal states)
  - Utility(s, p): value of terminal states, e.g., win (+1), lose (-1) and draw (0) in chess.
- Players use search tree to determine next move.

Partial Game Tree for Tic-Tac-Toe
The Tic-Tac-Toe Search Space

- Is this search space a tree or graph?
- What is the minimum search depth?
- What is the maximum search depth?
- What is the branching factor?

Optimal strategies

- Find the best strategy for MAX assuming an infallible MIN opponent.
- Assumption: Both players play optimally.
- Given a game tree, the optimal strategy can be determined by using the minimax value of each node:

\[
\text{MINIMAX}(s) = \begin{cases} 
\text{UTILITY}(s) & \text{if } s \text{ is a terminal} \\
\max_{a \in \text{Actions}(s)} \text{MINIMAX}(& \text{RESULT}(s,a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\
\min_{a \in \text{Actions}(s)} \text{MINIMAX}(& \text{RESULT}(s,a)) & \text{if } \text{PLAYER}(s) = \text{MIN}
\end{cases}
\]

Two-Ply Game Tree

Definition: ply = turn of a two-player game

The minimax value at a min node is the minimum of backed-up values, because your opponent will do what's best for them (and worst for you).

Minimax maximizes the worst-case outcome for max.
Minimax Algorithm

function MINIMAX-DECISION(state) returns an action
return arg max \ A \ \in \ Actions(state) \ \min \ VALUE(Result(state,a))

function MIN-VALUE(state) returns a utility value
if TERMINAL-TEST(state) then return UTILITY(state)
v ← \infty
for a in ACTIONS(state) do
v ← MIN(v, MAX-VALUE(Result(state,a)))
return v

function MAX-VALUE(state) returns a utility value
if TERMINAL-TEST(state) then return UTILITY(state)
v ← -\infty
for a in ACTIONS(state) do
v ← max(v, MIN-VALUE(Result(state,a)))
return v

Properties of Minimax

- Minimax explores tree using DFS.
- Therefore:
  - Time complexity: \(O(b^m)\)
  - Space complexity: \(O(bm)\)

Problem of minimax search

- Number of game states is exponential in the number of moves.
  - Solution: Do not examine every node
  - \(\Rightarrow\) Alpha-beta pruning
    - Remove branches that do not influence final decision
    - General idea: you can bracket the highest/lowest value at a node, even before all its successors have been evaluated

Alpha-Beta Pruning

- \(\alpha\): the highest (i.e. best for Max) value possible
- \(\beta\): the lowest (i.e. best for Min) value possible
- initially \(\alpha\) and \(\beta\) are \((-\infty, \infty)\).

Alpha-Beta Example

\(-\infty, +\infty\)

Range of possible values

\(-\infty, +3\)

Alpha-Beta Example (continued)
This node is worse for MAX.
**Alpha-Beta Example (continued)**

![Diagram](image)

**Alpha-Beta Algorithm**

```plaintext
function ALPHA-BETA-SEARCH(state) returns an action
return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
if TERMINAL-TEST(state) then return UTILITY(state)
v ← -∞
for each a in ACTIONS(state) do
  v ← MAX(v, MIN-VALUE(RESULT(state, a), α, β))
  if v ≥ β then return v
  α ← MAX(α, v)
return v

function MIN-VALUE(state, α, β) returns a utility value
if TERMINAL-TEST(state) then return UTILITY(state)
v ← +∞
for each a in ACTIONS(state) do
  v ← MIN(v, MAX-VALUE(RESULT(state, a), α, β))
  if v ≤ α then return v
  β ← MIN(β, v)
return v
```

**Alpha-beta pruning**

- When enough is known about a node n, it can be pruned.

**Final Comments about Alpha-Beta Pruning**

- Pruning does not affect final results
- Entire subtrees can be pruned, not just leaves.
- Good move ordering improves effectiveness of pruning
- With “perfect ordering,” time complexity is O(b^\(m/2\))
  - Effective branching factor of \(\sqrt{b}\)
  - Consequence: alpha-beta pruning can look twice as deep as minimax in the same amount of time

**Is this practical?**

- Minimax and alpha-beta pruning still have exponential complexity.
- May be impractical within a reasonable amount of time.
- SHANNON (1950):
  - Terminate search at a lower depth
  - Apply heuristic evaluation function EVAL instead of the UTILITY function
Cutting off search

- Change:
  - if TERMINAL-TEST(state) then return UTILITY(state)
  - if CUTOFF-TEST(state, depth) then return EVAL(state)
- Introduces a fixed-depth limit depth
  - Selected so that the amount of time will not exceed what the rules of the game allow.
  - When cutoff occurs, the evaluation is performed.

Heuristic EVAL

- Idea: produce an estimate of the expected utility of the game from a given position.
- Performance depends on quality of EVAL.
- Requirements:
  - EVAL should order terminal-nodes in the same way as UTILITY.
  - Fast to Compute.
  - For non-terminal states the EVAL should be strongly correlated with the actual chance of winning.

Heuristic EVAL example

Heuristic EVAL example

Heuristic EVAL example

Heuristic EVAL example

How good are computers…

- Let’s look at the state of the art computer programs that play games such as chess, checkers, othello, go…

Checkers

- Chinook: the first program to win the world champion title in a competition against a human (1994)
Chess

- 1997: Deep Blue wins a 6-game match against Garry Kasparov
- Searches using iterative deepening alpha-beta; evaluation function has over 8000 features; opening book of 4000 positions; endgame database
- FRITZ plays world champion, Vladimir Kramnik; wins 6-game match.

Othello

- The best Othello computer programs can easily defeat the best humans (e.g. Logistello, 1997).

Go

- Go: humans still much better!

Games that include chance

- Possible moves (5-10,5-11), (5-11,19-24),(5-10,10-16) and (5-11,11-16)
- [1,1],…,[6,6] probability 1/36, all others - 1/18
- Can not calculate definite minimax value, only expected value

Expected minimax value

\[
\text{EXPECTMINIMAX}(s) = \begin{cases} 
\text{UTILITY}(s) & \text{if } s \text{ is a terminal} \\
\max_a \text{EXPECTMINIMAX}(\text{RESULT}(s,a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\
\min_a \text{EXPECTMINIMAX}(\text{RESULT}(s,a)) & \text{if } \text{PLAYER}(s) = \text{MIN} \\
\sum_r \Pr(r) \text{EXPECTMINIMAX}(\text{RESULT}(s,r)) & \text{if } \text{PLAYER}(s) = \text{CHANCE} 
\end{cases}
\]

r is a chance event (e.g., a roll of the dice). These equations can be propagated recursively in a similar way to the MINIMAX algorithm.
TD-Gammon (Tesauro, 1994)

World class program based on a combination of reinforcement Learning, neural networks and alpha-beta pruning to 3 plies.

Move analyses by TD-Gammon have lead to some changes in accepted strategies.


Summary

- Games are fun
- They illustrate several important points about AI
  - Perfection is (usually) unattainable -> approximation
  - Uncertainty constrains the assignment of values to states