Adversarial Search and Game Playing

Russell and Norvig, Chapter 5
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>Types of Information</th>
<th>Deterministic</th>
<th>Chance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perfect information</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
Partial Game Tree for Tic-Tac-Toe
COMPLETE MAP OF OPTIMAL TIC-TAC-TOE MOVES

YOUR MOVE IS GIVEN BY THE POSITION OF THE LARGEST RED SYMBOL ON THE GRID, WHEN YOUR OPPONENT PICKS A MOVE, ZOOM IN ON THE REGION OF THE GRID WHERE THEY WENT. REPEAT.

MAP FOR X:
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?
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.
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:

\[
\text{MINIMAX}(s) =
\begin{align*}
&\text{UTILITY}(s) &\text{If } s \text{ is a terminal} \\
&\max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) &\text{If } \text{PLAYER}(s) = \text{MAX} \\
&\min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) &\text{If } \text{PLAYER}(s) = \text{MIN}
\end{align*}
\]
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).
Two-Ply Game Tree

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

function MINIMAX-DECISION(state) returns an action
return arg max \( a \in \text{Actions}(s) \) MIN-VALUE(RESULT(state,a))

function MAX-VALUE(state) returns a utility value
if TERMINAL-TEST(state) then return UTILITY(state)
v ← -\( \infty \)
for each a in ACTIONS(state) do
\[ v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(state,a))) \]
return v

function MIN-VALUE(state) returns a utility value
if TERMINAL-TEST(state) then return UTILITY(state)
v ← \( \infty \)
for a in ACTIONS(state) do
\[ v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(state,a))) \]
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
  - ==> 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 Example

Range of possible values

\([-\infty, +\infty]\)
Alpha-Beta Example (continued)
Alpha-Beta Example (continued)
Alpha-Beta Example (continued)

![Diagram of Alpha-Beta pruning]

- MAX
- MIN

Values:
- [3, 3]
- [3, +∞]
Alpha-Beta Example (continued)

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

- $\alpha$: the best value for MAX (i.e. highest) along a path from the root
- $\beta$: the best value for MIN (i.e. lowest) along a path from the root
- initially $\alpha$ and $\beta$ are $(\neg\infty, \infty)$. 
function ALPHA-BETA-SEARCH(state) returns an action

\[ v \leftarrow \text{MAX-VALUE}(state, -\infty, +\infty) \]

return the action in ACTIONS(state) with value \( v \)

function MAX-VALUE(state, \( \alpha \), \( \beta \)) returns a utility value

if TERMINAL-TEST(state) then return UTILITY(state)

\[ v \leftarrow -\infty \]

for each \( a \) in ACTIONS(state) do

\[ v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(<state,a>, \alpha, \beta)) \]

if \( v \geq \beta \) then return \( v \)

\[ \alpha \leftarrow \text{MAX}(\alpha, v) \]

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

$$\text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

In chess:
$$w_1 \text{ material} + w_2 \text{ mobility} + w_3 \text{ king safety} + w_4 \text{ center control} + \ldots$$
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)
Chinook

Components of Chinook:
- Search (variant of alpha-beta). Search space has $10^{20}$ states.
- Evaluation function
- Endgame database (for all states with 4 vs. 4 pieces; roughly 444 billion positions).
- Opening book - a database of opening moves

Chinook can determine the final result of the game within the first 10 moves.

2007: Checkers is solved. Perfect play leads to a draw.

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; end game database.
  - FRITZ plays world champion, Vladimir Kramnik; wins 6-game match.
The best Othello computer programs can easily defeat the best humans (e.g. Logistello, 1997).
Recently, AlphaGo and its subsequent improvements defeated professional human players.
Games that include chance

- Possible moves (5-10,5-11), (5-11,19-24),(5-10,10-16) and (5-11,11-16)
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

\[
\text{EXPECTIMINIMAX}(s) =
\begin{align*}
\text{UTILITY}(s) & \quad \text{If } s \text{ is a terminal} \\
\max_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \quad \text{If } \text{PLAYER}(S) = \text{MAX} \\
\min_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \quad \text{If } \text{PLAYER}(S) = \text{MIN} \\
\sum_r P(r) \text{EXPECTIMINIMAX}(\text{RESULT}(s,r)) & \quad \text{If } \text{PLAYER}(S) = \text{CHANCE}
\end{align*}
\]

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 led to some changes in accepted strategies.

Summary

- Games are fun
- They illustrate several important points about AI
  - Perfection is (usually) unattainable -> approximation