ADVERSARIAL SEARCH

Chapter 5

“... every game of skill is susceptible of being played by an automaton.”
from Charles Babbage, The Life of a Philosopher, 1832.
Outline

◇ Games

◇ Perfect play
   – minimax decisions
   – $\alpha - \beta$ pruning

◇ Resource limits and approximate evaluation

◇ Games of chance

◇ Games of imperfect information
“Unpredictable” opponent ⇒ solution is a **strategy**
specifying a move for every possible opponent reply

Time limits ⇒ unlikely to find goal, must approximate

**Plan of attack:**

- Computer considers possible lines of play (Babbage, 1846)
- Algorithm for perfect play (Zermelo, 1912; Von Neumann, 1944)
- Finite horizon, approximate evaluation (Zuse, 1945; Wiener, 1948; Shannon, 1950)
- First chess program (Turing, 1951)
- Machine learning to improve evaluation accuracy (Samuel, 1952–57)
- Pruning to allow deeper search (McCarthy, 1956)
<table>
<thead>
<tr>
<th>Types of games</th>
<th>Deterministic</th>
<th>Chance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perfect information</td>
<td>chess, checkers, go, othello</td>
<td>backgammon monopoly</td>
</tr>
<tr>
<td>Imperfect information</td>
<td>battleships, blind tictactoe</td>
<td>bridge, poker, scrabble nuclear war</td>
</tr>
</tbody>
</table>
Game tree (2-player, deterministic, turns)

MAX (X)

MIN (O)

MAX (X)

MIN (O)

TERMINAL

Utility

Utility

Utility

Utility
Minimax

Perfect play for deterministic, perfect-information games

Idea: choose move to position with highest minimax value
      = best achievable payoff against best play

E.g., 2-ply game:

```
MAX

A_1
   /\    /
A_2  A_3
/
A_11 A_12 A_13

MIN

3
   /\    /
2  2
/
A_21 A_22 A_23
/
A_31 A_32 A_33
```

Chapter 6  6
Minimax algorithm

**function** Minimax-Decision(state) **returns** an action

**inputs:** state, current state in game

**return** the a in ACTIONS(state) maximizing Min-Value(Result(a, state))

**function** Max-Value(state) **returns** a utility value

**if** Terminal-Test(state) **then** return Utility(state)

v ← −∞

**for** a, s in Successors(state) **do** v ← Max(v, Min-Value(s))

**return** v

**function** Min-Value(state) **returns** a utility value

**if** Terminal-Test(state) **then** return Utility(state)

v ← ∞

**for** a, s in Successors(state) **do** v ← Min(v, Max-Value(s))

**return** v
Properties of minimax

Complete??
Properties of minimax

Complete?? Only if tree is finite (chess has specific rules for this).

NB a finite strategy can exist even in an infinite tree!

Optimal??
Properties of minimax

Complete?? Yes, if tree is finite (chess has specific rules for this)

Optimal?? Yes, against an optimal opponent. Otherwise??

Time complexity??
Properties of minimax

Complete? Yes, if tree is finite (chess has specific rules for this)

Optimal? Yes, against an optimal opponent. Otherwise?

Time complexity? $O(b^m)$

Space complexity??
Properties of minimax

Complete?? Yes, if tree is finite (chess has specific rules for this)

Optimal?? Yes, against an optimal opponent. Otherwise??

Time complexity?? $O(b^m)$

Space complexity?? $O(bm)$ (depth-first exploration)

For chess, $b \approx 35$, $m \approx 100$ for “reasonable” games
  $\Rightarrow$ exact solution completely infeasible

But do we need to explore every path?
$\alpha-\beta$ pruning example

\[\begin{array}{c}
\text{MAX} \\
\text{MIN} \\
3 \\
12 \\
8 \\
\end{array} \ 
\begin{array}{c}
\geq 3 \\
\end{array}\]
\( \alpha - \beta \) pruning example

MAX

MIN

3 12 8

\[ \geq 3 \]

\[ \leq 2 \]

3 2 X X
\(\alpha-\beta\) pruning example

```
MAX

MIN

3
12
8
2
14
```

\[\geq 3\]

\[\leq 2\]

\[\leq 14\]
\( \alpha-\beta \) pruning example

\[
\begin{array}{c}
\text{MAX} \\
\text{MIN}
\end{array}
\]

\[
\begin{array}{cccc}
3 & 12 & 8 & 2 \\
\geq 3 & \leq 2 & \times & \leq 5
\end{array}
\]

Chapter 6
\( \alpha - \beta \) pruning example
Why is it called $\alpha - \beta$?

$\alpha$ is the best value (to MAX) found so far off the current path
If $V$ is worse than $\alpha$, MAX will avoid it $\Rightarrow$ prune that branch
Define $\beta$ similarly for MIN
Properties of $\alpha-\beta$

Pruning does not affect final result

Good move ordering improves effectiveness of pruning

With “perfect ordering,” time complexity $= O(b^{m/2})$

$\Rightarrow$ doubles solvable depth

A simple example of the value of reasoning about which computations are relevant (a form of metareasoning)

Unfortunately, $35^{50}$ is still impossible!
Resource limits

Standard approach:

- Use **Cutoff-Test** instead of **Terminal-Test**
  e.g., depth limit (perhaps add quiescence search)

- Use **Eval** instead of **Utility**
  i.e., evaluation function that estimates desirability of position

Suppose we have 100 seconds, explore $10^4$ nodes/second
  \[ \Rightarrow 10^6 \text{ nodes per move} \approx 35^{8/2} \]
  \[ \Rightarrow \alpha-\beta \text{ reaches depth 8} \Rightarrow \text{pretty good chess program} \]
Evaluation functions

For chess, typically linear weighted sum of features

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

e.g., $w_1 = 9$ with

$f_1(s) = \text{(number of white queens)} - \text{(number of black queens)}$, etc.
Define $X_n$ as the number of rows, columns, or diagonals with exactly $n$ $X$'s and no $O$'s. Similarly, $O_n$ is the number of rows, columns, or diagonals with exactly $n$ $O$'s and no $X$'s.

The utility function assigns +1 to any position with $X_3 = 1$ and -1 for any position with $O_3 = 1$. All other terminal positions have utility 0.

For non-terminal positions, we use a linear evaluation function defined as $\text{Eval}(s) = 3X_2(s) + X_1(s) - (3O_2(s) + O_1(s))$

a) Approximately how many games of tic-tac-toe are there?
Exercise – Tic-tac-toe

b) What does the game tree look like (taking symmetry into account)?
Digression: Exact values don’t matter

Behaviour is preserved under any *monotonic* transformation of \texttt{Eval}

Only the order matters:
- payoff in deterministic games acts as an *ordinal utility* function
How to achieve a good game of chess?

- Extensively tuned **evaluation function**
- Cutoff test with **quiescence search**
- Large **transposition table** [i.e., hash of previously seen positions, saved for re-use]
- Use of **alpha-beta**, with extra pruning
- Large database of **optimal opening and endgame moves**
- **Fast computer!**
Exercise – Prove correctness of $\alpha$-$\beta$

- Question is whether to prune $n_j$, which is a max-node and descendent of $n_1$
- Basic idea is to prune it iff the minimax value of $n_1$ can be shown to be independent of the value of $n_j$
- Node $n_1$ takes on the minimum value among its children $n_1 = \min(n_2, n_{2_1}, \ldots, n_{2_{b_2}})$. Find a similar expression for $n_2$ and hence an expression for $n_1$ in terms of $n_j$.  

\[ n_1 = \min(n_2, n_{2_1}, \ldots, n_{2_{b_2}}) \]