Game playing

Chapter 6, Sections 1–8

Outline

◊ Perfect play
◊ Resource limits
◊ $\alpha$–$\beta$ pruning
◊ 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)

Types of games

<table>
<thead>
<tr>
<th>deterministic</th>
<th>chance</th>
</tr>
</thead>
<tbody>
<tr>
<td>perfect information</td>
<td>imperfect information</td>
</tr>
<tr>
<td>chess, checkers, go, othello</td>
<td>backgammon monopoly</td>
</tr>
<tr>
<td>bridge, poker, scrabble nuclear war</td>
<td></td>
</tr>
</tbody>
</table>
Game tree (2-player, deterministic, turns)

Max (X)

Min (O)

Max (X)

Min (O)

Terminal

Utility

-1 0 +1

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

Min

A11 A12 A13

A21 A22 A23

A31 A32 A33

3 12 8 2 4 6 14 5 2
Minimax algorithm

function MINIMAX-DECISION(state, game) returns an action

    action, state ← the a, s in SUCCESSORS(state)
    such that MINIMAX-VALUE(s, game) is maximized
    return action

function MINIMAX-VALUE(state, game) returns a utility value

    if TERMINAL-TEST(state) then
        return UTILITY(state)
    else if MAX is to move in state then
        return the highest MINIMAX-VALUE of SUCCESSORS(state)
    else
        return the lowest MINIMAX-VALUE of SUCCESSORS(state)

Properties of minimax

Complete??
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**?

For chess, $b \approx 35$, $m \approx 100$ for “reasonable” games

$\Rightarrow$ exact solution completely infeasible
Resource limits

Suppose we have 100 seconds, explore $10^4$ nodes/second
$\Rightarrow 10^6$ nodes per move

Standard approach:

- \textit{cutoff test}
  
e.g., depth limit (perhaps add \textit{quiescence search})

- \textit{evaluation function}
  
equal estimated desirability of position

Evaluation functions

\begin{itemize}
  \item \textbf{Black to move}
    \begin{itemize}
      \item White slightly better
    \end{itemize}
  \item \textbf{White to move}
    \begin{itemize}
      \item Black winning
    \end{itemize}
\end{itemize}

For chess, typically \textit{linear} weighted sum of \textit{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

\[ f_1(s) = (\text{number of white queens}) - (\text{number of black queens}), \text{ etc.} \]
Digression: Exact values don’t matter

Behaviour is preserved under any monotonic transformation of Eval.

Only the order matters:
- payoff in deterministic games acts as an ordinal utility function.

Cutting off search

**MINIMAXCUTOFF** is identical to **MINIMAXVALUE** except
1. **TERMINAL**? is replaced by **CUTOFF**?
2. **UTILITY** is replaced by **Eval**.

Does it work in practice?

\[ b^m = 10^6, \quad b = 35 \quad \Rightarrow \quad m = 4 \]

4-ply lookahead is a hopeless chess player!
- 4-ply \(\approx\) human novice
- 8-ply \(\approx\) typical PC, human master
- 12-ply \(\approx\) Deep Blue, Kasparov
\section*{\textit{\alpha-\beta} pruning example}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{example_diagram}
\caption{\textit{\alpha-\beta} pruning example diagram}
\end{figure}
Pruning *does not* affect final result

Good move ordering improves effectiveness of pruning

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

⇒ *doubles* depth of search
⇒ can easily reach depth 8 and play good chess

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

The $\alpha-\beta$ algorithm

function \textsc{Alpha-Beta-Search}(state, game) \textbf{returns} an action
\hspace{1em} action, state $\leftarrow$ the a, $s$ in $\text{Successors}[\text{game}](\text{state})$
\hspace{1em} such that $\text{Min-Value}(s, \text{game}, -\infty, +\infty)$ is maximized
\hspace{1em} return action

function \textsc{Max-Value}(state, game, $\alpha$, $\beta$) \textbf{returns} the minimax value of $state$
\hspace{1em} if \textsc{Cutoff-Test}(state) then return \textsc{Eval}(state)
\hspace{1em} for each $s$ in \text{Successors}(state) do
\hspace{2em} $\alpha \leftarrow \max(\alpha, \text{Min-Value}(s, \text{game}, \alpha, \beta))$
\hspace{2em} if $\alpha \geq \beta$ then return $\beta$
\hspace{1em} return $\alpha$

function \textsc{Min-Value}(state, game, $\alpha$, $\beta$) \textbf{returns} the minimax value of $state$
\hspace{1em} if \textsc{Cutoff-Test}(state) then return \textsc{Eval}(state)
\hspace{1em} for each $s$ in \text{Successors}(state) do
\hspace{2em} $\beta \leftarrow \min(\beta, \text{Max-Value}(s, \text{game}, \alpha, \beta))$
\hspace{2em} if $\beta \leq \alpha$ then return $\alpha$
\hspace{1em} return $\beta$
Deterministic games in practice

Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.


Othello: human champions refuse to compete against computers, who are too good.

Go: human champions refuse to compete against computers, who are too bad. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.

Nondeterministic games: backgammon

![Backgammon Board](attachment:backgammon_board.png)
Nondeterministic games in general

In nondeterministic games, chance introduced by dice, card-shuffling

Simplified example with coin-flipping:

```
MAX

CHANCE

2

MIN

3

0.5

4

0.5

5

0.5

-1

0

-2

0.5

-2

0

-2

0.5

-2
```

Algorithm for nondeterministic games

EXPECTIMINIMAX gives perfect play

Just like MINIMAX, except we must also handle chance nodes:

```
if state is a MAX node then
    return the highest EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
if state is a MIN node then
    return the lowest EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
if state is a chance node then
    return average of EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
```

...
Pruning in nondeterministic game trees

A version of $\alpha$-$\beta$ pruning is possible:

![Diagram of a game tree with pruning applied]

29

Pruning in nondeterministic game trees

A version of $\alpha$-$\beta$ pruning is possible:

![Diagram of a game tree with pruning applied]

30
A version of $\alpha$-$\beta$ pruning is possible:

![Diagram of a nondeterministic game tree](image)

Pruning in nondeterministic game trees
A version of $\alpha$-$\beta$ pruning is possible:

Pruning in nondeterministic game trees

A version of $\alpha$-$\beta$ pruning is possible:
Pruning in nondeterministic game trees

A version of $\alpha$-$\beta$ pruning is possible:
More pruning occurs if we can bound the leaf values.
More pruning occurs if we can bound the leaf values
More pruning occurs if we can bound the leaf values.
**Nondeterministic games in practice**

Dice rolls increase: 21 possible rolls with 2 dice
Backgammon ≈ 20 legal moves (can be 6,000 with 1-1 roll)

\[
\text{depth } 4 = 20 \times (21 \times 20)^3 \approx 1.2 \times 10^9
\]

As depth increases, probability of reaching a given node shrinks
⇒ value of lookahead is diminished

\(\alpha-\beta\) pruning is much less effective

TDGAMMOM uses depth-2 search + very good EVAL
≈ world-champion level

---

**Digression: Exact values DO matter**

<table>
<thead>
<tr>
<th>DICE</th>
<th>MIN</th>
<th>MAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>.9</td>
<td>22</td>
</tr>
<tr>
<td>.1</td>
<td>.9</td>
<td>33</td>
</tr>
<tr>
<td>.9</td>
<td>.1</td>
<td>11</td>
</tr>
<tr>
<td>.1</td>
<td>.9</td>
<td>4</td>
</tr>
</tbody>
</table>

Behaviour is preserved only by *positive linear* transformation of EVAL
Hence EVAL should be proportional to the expected payoff
Games of imperfect information

E.g., card games, where opponent’s initial cards are unknown
Typically we can calculate a probability for each possible deal
Seems just like having one big dice roll at the beginning of the game
Idea: compute the minimax value of each action in each deal,
then choose the action with highest expected value over all
deals
Special case: if an action is optimal for all deals, it’s optimal
GIB, current best bridge program, approximates this idea by
1) generating 100 deals consistent with bidding information
2) picking the action that wins most tricks on average

Example
Four-card bridge/whist/ hearts hand, MAX to play first
Example

Four-card bridge/whist/hearts hand, MAX to play first

MAX
\[ \begin{array}{c|c|c|c}
| 5\spadesuit | 6\spadesuit | 7\spadesuit | 8\spadesuit |
| 6\diamondsuit | 2\spadesuit & 9\spadesuit & 3\spadesuit |
| 4\diamondsuit | 3\spadesuit & 4\diamondsuit | 3\spadesuit |
\end{array} \]

MIN
\[ \begin{array}{c|c|c|c}
| 2\heartsuit & 8\heartsuit & 7\heartsuit & 9\heartsuit |
| 6\diamondsuit & 6\diamondsuit & 8\diamondsuit & 7\diamondsuit |
| 4\diamondsuit & 4\diamondsuit & 4\diamondsuit & 4\diamondsuit |
\end{array} \]

0

Example

Four-card bridge/whist/hearts hand, MAX to play first

MAX
\[ \begin{array}{c|c|c|c}
| 5\spadesuit | 6\spadesuit | 7\spadesuit | 8\spadesuit |
| 6\diamondsuit | 2\spadesuit & 9\spadesuit & 3\spadesuit |
| 4\diamondsuit | 3\spadesuit & 4\diamondsuit | 3\spadesuit |
\end{array} \]

MIN
\[ \begin{array}{c|c|c|c}
| 2\heartsuit & 8\heartsuit & 7\heartsuit & 9\heartsuit |
| 6\diamondsuit & 6\diamondsuit & 8\diamondsuit & 7\diamondsuit |
| 4\diamondsuit & 4\diamondsuit & 4\diamondsuit & 4\diamondsuit |
\end{array} \]

0

MAX
\[ \begin{array}{c|c|c|c}
| 5\spadesuit | 6\spadesuit | 7\spadesuit | 8\spadesuit |
| 6\diamondsuit | 2\spadesuit & 9\spadesuit & 3\spadesuit |
| 4\diamondsuit | 3\spadesuit & 4\diamondsuit | 3\spadesuit |
\end{array} \]

MIN
\[ \begin{array}{c|c|c|c}
| 2\heartsuit & 8\heartsuit & 7\heartsuit & 9\heartsuit |
| 6\diamondsuit & 6\diamondsuit & 8\diamondsuit & 7\diamondsuit |
| 4\diamondsuit & 4\diamondsuit & 4\diamondsuit & 4\diamondsuit |
\end{array} \]

0

MAX
\[ \begin{array}{c|c|c|c}
| 5\spadesuit | 6\spadesuit | 7\spadesuit | 8\spadesuit |
| 6\diamondsuit | 2\spadesuit & 9\spadesuit & 3\spadesuit |
| 4\diamondsuit | 3\spadesuit & 4\diamondsuit | 3\spadesuit |
\end{array} \]

MIN
\[ \begin{array}{c|c|c|c}
| 2\heartsuit & 8\heartsuit & 7\heartsuit & 9\heartsuit |
| 6\diamondsuit & 6\diamondsuit & 8\diamondsuit & 7\diamondsuit |
| 4\diamondsuit & 4\diamondsuit & 4\diamondsuit & 4\diamondsuit |
\end{array} \]

-0.5

MAX
\[ \begin{array}{c|c|c|c}
| 5\spadesuit | 6\spadesuit | 7\spadesuit | 8\spadesuit |
| 6\diamondsuit | 2\spadesuit & 9\spadesuit & 3\spadesuit |
| 4\diamondsuit | 3\spadesuit & 4\diamondsuit | 3\spadesuit |
\end{array} \]

MIN
\[ \begin{array}{c|c|c|c}
| 2\heartsuit & 8\heartsuit & 7\heartsuit & 9\heartsuit |
| 6\diamondsuit & 6\diamondsuit & 8\diamondsuit & 7\diamondsuit |
| 4\diamondsuit & 4\diamondsuit & 4\diamondsuit & 4\diamondsuit |
\end{array} \]

-0.5
Commonsense example

Road A leads to a small heap of gold pieces
Road B leads to a fork:
    take the left fork and you’ll find a mound of jewels;
    take the right fork and you’ll be run over by a bus.
Commonsense example

Road A leads to a small heap of gold pieces
Road B leads to a fork:
    take the left fork and you’ll find a mound of jewels;
    take the right fork and you’ll be run over by a bus.

Road A leads to a small heap of gold pieces
Road B leads to a fork:
    take the left fork and you’ll be run over by a bus;
    take the right fork and you’ll find a mound of jewels.

Road A leads to a small heap of gold pieces
Road B leads to a fork:
    guess correctly and you’ll find a mound of jewels;
    guess incorrectly and you’ll be run over by a bus.

Proper analysis

* Intuition that the value of an action is the average of its values in all actual states is WRONG

With partial observability, value of an action depends on the information state or belief state the agent is in

Can generate and search a tree of information states

Leads to rational behaviors such as
    ◇ Acting to obtain information
    ◇ Signalling to one’s partner
    ◇ Acting randomly to minimize information disclosure
Summary

Games are fun to work on! (and dangerous)
They illustrate several important points about AI
◊ perfection is unattainable ⇒ must approximate
◊ good idea to think about what to think about
◊ uncertainty constrains the assignment of values to states
Games are to AI as grand prix racing is to automobile design