Game playing

Chapter 5, Sections 1–5

Outline

- Perfect play
- Resource limits
- $\alpha$$\beta$ pruning
- Games of chance
- Games of imperfect information

Game tree (2-player, deterministic, turns)

Types of games

<table>
<thead>
<tr>
<th></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>bridge, poker, scrabble nuclear war</td>
<td></td>
</tr>
</tbody>
</table>

Games vs. search problems

"Unpredictable" opponent $\Rightarrow$ solution is a strategy specifying a move for every possible opponent reply

Time limits $\Rightarrow$ 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, 1943; 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)

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-\$\$ game:

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

\[
\begin{array}{c}
A_1 \\
A_2 \\
A_3 \\
A_4 \\
A_5 \\
A_6 \\
A_7 \\
A_8 \\
\end{array}
\]

\[
\begin{array}{c}
B_1 \\
B_2 \\
B_3 \\
B_4 \\
B_5 \\
B_6 \\
B_7 \\
B_8 \\
\end{array}
\]

\[
\begin{array}{c}
C_1 \\
C_2 \\
C_3 \\
C_4 \\
C_5 \\
C_6 \\
C_7 \\
C_8 \\
\end{array}
\]

\[
\begin{array}{c}
D_1 \\
D_2 \\
D_3 \\
\end{array}
\]

\[
\begin{array}{c}
E_1 \\
E_2 \\
E_3 \\
\end{array}
\]
Minimax algorithm

function MinimaxDecision(state, game) returns an action
action, state = the a in SUCCESSORS(state)
such that Minimax-Value(state) 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?? Yes, if tree is finite (chess has specific rules for this)
Optimal?? Yes, against an optimal opponent. Otherwise??
Time complexity??

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?? $O(b^n)$
Space complexity?? $O(bn)$ (depth-first exploration)

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

Properties of minimax

Complete?? Yes, if tree is finite (chess has specific rules for this)
Optimal?? Yes, against an optimal opponent. Otherwise??
Time complexity??
Space complexity??
Resource limits

Suppose we have 100 seconds, explore 10^4 nodes/second
⇒ 10^7 nodes per move

Standard approach:
- **cutoff test**
  e.g., depth limit (perhaps add quiescence search)
- **evaluation function**
  - estimated desirability of position

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?
- \( b^n = 10^6 \), \( b = 35 \) ⇒ \( n = 4 \)
- 4-ply lookahead is a hopeless chess player!
- 8-ply = typical PC, human master
- 12-ply = Deep Blue, Kasparov

Evaluation functions

For chess, typically linear weighted sum of features

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

e.g., \( w_1 = 9 \) with
\( f_1(s) = \text{(number of white queens)} - \text{(number of black queens)} \), etc.

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
Properties of α−β

Pruning does not affect final result.

Good move ordering improves effectiveness of pruning.

With "perfect ordering," time complexity = \(O(b^d/2)\)

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

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

Why is it called α−β?

\(\alpha\) is the best value (to \(\MAX\)) found so far off the current path.

If \(V\) is worse than \(\alpha\), \(\MAX\) will avoid it ⇒ prune that branch.

Define \(\beta\) similarly for \(\MIN\).

The α−β algorithm

```
function ALPHA-BETA-SEARCH(state, game) returns an action
  action, state := \(\alpha\) in SUCCESSORS(game(state))
  such that MIN-VALUE(game(state), -\(\infty\), +\(\infty\)) is maximized
  return action

function MIN-VALUE(state, game, \(\alpha\), \(\beta\)) returns the minimax value of state
  if CUTOFF-TEST(state) then return EVAL(state)
  for each \(s\) in SUCCESSORS(state) do
    \(\beta := \min(\beta, \text{MIN-VALUE}(s, \text{game}, \alpha, \beta))\)
    if \(\alpha \geq \beta\) then return \(\beta\)
  return \(\alpha\)

function MAX-VALUE(state, game, \(\alpha\), \(\beta\)) returns the maximax value of state
  if CUTOFF-TEST(state) then return EVAL(state)
  for each \(s\) in SUCCESSORS(state) do
    \(\alpha := \max(\alpha, \text{MAX-VALUE}(s, \text{game}, \alpha, \beta))\)
    if \(\alpha \geq \beta\) then return \(\alpha\)
  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, $d > 300$, so most programs use pattern knowledge bases to suggest plausible moves.

Algorithm for nondeterministic games

**EXPECTIMINAX** gives perfect play
Just like **MINIMAX**, except we must also handle chance nodes:

\[
\begin{align*}
\text{if state is a Max node then} & \quad \text{return the highest ExpectedMinimax-Value of Successors(state)} \\
\text{if state is a Min node then} & \quad \text{return the lowest ExpectedMinimax-Value of Successors(state)} \\
\text{if state is a chance node then} & \quad \text{return average of ExpectedMinimax-Value of Successors(state)} \\
\end{align*}
\]

Nondeterministic games: backgammon

Pruning in nondeterministic game trees

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

\[
\begin{align*}
\text{MAX} & \quad \text{CHANCE} \\
3 & \quad 0.5 & \quad 0.5 & \quad 0.5 & \quad 0.5 \\
\text{MIN} & \quad 2 & \quad 4 & \quad 7 & \quad 4 & \quad 6 & \quad 0 & \quad 5 & \quad -2
\end{align*}
\]

Nondeterministic games in general

**Pruning in nondeterministic game trees**

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

\[
\begin{align*}
\text{MAX} & \quad \text{CHANCE} \\
3 & \quad 0.5 & \quad 0.5 & \quad 0.5 & \quad 0.5 \\
\text{MIN} & \quad 2 & \quad 4 & \quad 7 & \quad 4 & \quad 6 & \quad 0 & \quad 5 & \quad -2
\end{align*}
\]

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

Simplified example with coin-flipping:
Pruning contd.

More pruning occurs if we can bound the leaf values

Pruning contd.

More pruning occurs if we can bound the leaf values

Pruning contd.

More pruning occurs if we can bound the leaf values

Pruning contd.

More pruning occurs if we can bound the leaf values
Nondeterministic games in practice

Dice rolls increase 6: 21 possible rolls with 2 dice
Backgammon ≈ 20 legal moves (can be 6,000 with 1-1 roll)
depth 4 = 20 × (21 × 20)³ ≈ 1.2 × 10⁴

As depth increases, probability of reaching a given node shrinks
⇒ value of lookahead is diminished
α-β pruning is much less effective

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

Digression: Exact values DO matter

MAX

DICE

MIN

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.

GBR, 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.
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.

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

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.

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

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.