Game playing

Chapter 5, Sections 1–5

Outline

◊ Perfect play
◊ Resource limits
◊ α–β pruning
◊ Games of chance
Games vs. search problems

"Unpredictable" opponent ⇒ solution is a contingency plan

Time limits ⇒ unlikely to find goal, must approximate

Plan of attack:

- algorithm for perfect play (Von Neumann, 1944)
- finite horizon, approximate evaluation (Zuse, 1945; Shannon, 1950; Samuel, 1952–57)
- pruning to reduce costs (McCarthy, 1956)

Types of games

<table>
<thead>
<tr>
<th>perfect information</th>
<th>deterministic</th>
<th>chance</th>
</tr>
</thead>
<tbody>
<tr>
<td>chess, checkers, go, othello</td>
<td></td>
<td>backgammon</td>
</tr>
<tr>
<td></td>
<td></td>
<td>monopoly</td>
</tr>
<tr>
<td>imperfect information</td>
<td></td>
<td>bridge, poker,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>scrabble nuclear war</td>
</tr>
</tbody>
</table>
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

MIN
  3
  3
```

Minimax algorithm

```
function Minimax-Decision(game) returns an operator
  for each op in Operators[game] do
    Value[op] ← Minimax-Value(Apply(op, game), game)
  end
  return the op with the highest Value[op]

function Minimax-Value(state, game) returns a utility value
  if Terminal-Test(game)(state) then
    return Utility[game](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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Complete</strong></td>
<td>Yes, if tree is finite (chess has specific rules for this)</td>
</tr>
<tr>
<td><strong>Optimal</strong></td>
<td>Yes, against an optimal opponent. Otherwise</td>
</tr>
<tr>
<td><strong>Time complexity</strong></td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td><strong>Space complexity</strong></td>
<td>$O(bm)$ (depth-first exploration)</td>
</tr>
</tbody>
</table>

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:

- cutoff test
  - e.g., depth limit (perhaps add quiescence search)

- evaluation function
  - $= \text{estimated desirability of position}$

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**

![Game tree diagram]

Behaviour is preserved under any *monotonic* transformation of \( \text{Eval} \)
Only the order matters:
  payoff in deterministic games acts as an *ordinal utility* function

---

**Cutting off search**

**Minimax\text{Cutoff}** 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
\( \alpha-\beta \) pruning example

```
MAX

MIN

3
3 12 8
```

```
MAX

MIN

3
3 12 8
```

```
MAX

MIN

3
3 12 8

X
X
```
MAX

MIN

CS 580

15

CS 580

16
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* depth of search  
$\Rightarrow$ 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 $\text{MAX}$) found so far off the current path

If $V$ is worse than $\alpha$, $\text{MAX}$ will avoid it $\Rightarrow$ prune that branch

Define $\beta$ similarly for $\text{MIN}$

---

The $\alpha-\beta$ algorithm

```python
function Max-Value(state, game, $\alpha$, $\beta$) returns the minimax value of $state$
    inputs: $state$, current state in game
            $game$, game description
            $\alpha$, the best score for $\text{MAX}$ along the path to $state$
            $\beta$, the best score for $\text{MIN}$ along the path to $state$
    if Cutoff-Test($state$) then return Eval($state$)
    for each $s$ in Successors($state$) do
        $\alpha$ ← Max($\alpha$, Min-Value($s$, game, $\alpha$, $\beta$))
        if $\alpha$ ≥ $\beta$ then return $\beta$
    end
    return $\alpha$

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($\beta$, Max-Value($s$, game, $\alpha$, $\beta$))
        if $\beta$ ≤ $\alpha$ then return $\alpha$
    end
    return $\beta$
```

Basically $\text{MINIMAX} + \text{keep track of } \alpha, \beta + \text{prune}$
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

E.g, in backgammon, the dice rolls determine the legal moves

Simplified example with coin-flipping instead of dice-rolling:
**Algorithm for nondeterministic games**

**EXPECTIMINIMAX** gives perfect play

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

\[
\text{if state is a chance node then}
\]

\[
\text{return average of} \ \text{EXPECTIMINIMAX-VALUE of SUCCESSORS(state)}
\]

\[
\text{A version of } \alpha-\beta \text{ pruning is possible}
\]

but only if the leaf values are bounded. **Why??**

---

**Nondeterministic games in practice**

Dice rolls increase \( b \): 21 possible rolls with 2 dice

Backgammon \( \approx 20 \) 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

\( \Rightarrow \) value of lookahead is diminished

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

**TDGAMMON** uses depth-2 search + very good **Eval**

\( \approx \) world-champion level
**Digression: Exact values DO matter**

Behaviour is preserved only by *positive linear* transformation of \( \text{Eval} \)
Hence \( \text{Eval} \) should be proportional to the expected payoff

---

**Summary**

Games are fun to work on! (and dangerous)
They illustrate several important points about AI
◊ perfection is unattainable \( \Rightarrow \) 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