Chapter 6

Outline

♦ Games
♦ Perfect play
  – minimax decisions
  – \( \alpha - \beta \) pruning
♦ Resource limits and approximate evaluation
♦ Games of chance
♦ Games of imperfect information

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)
• Machine learning to improve evaluation accuracy (Zentner, 1992-97)

Game vs. search problems

"Unpredictable" opponent
⇒ solution is a strategy specifying a move for every possible opponent reply

Game playing

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)
• Machine learning to improve evaluation accuracy (Zentner, 1992-97)

Game playing

Types of games

Determinate

Imperfect Information

Bridge, poker, scrabble

Backgammon

Go, checkers, chess

Nuclear war

Monopoly

Battlehip

Impartial Information

Generations

Chess, checkers,
Minimax algorithm

**Minimax-Decision**

(state)

returns an action

**Inputs:**

- state, current state in game

**Return:**

- the action 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**

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

For chess, `b ≈ 35`, `m ≈ 100` for “reasonable” games

⇒ exact solution completely infeasible

But do we need to explore every path?

For chess, it is ≈ 10^12, for “reasonable” games
Why is it called \( \alpha - \beta \) pruning?

\( \alpha \) is the best value (to \textsc{max}) found so far on the current path.

If \( V \) is worse than \( \alpha \), \textsc{max} will avoid it. \( \Rightarrow \) prune that branch.

Define \( \beta \) similarly for \textsc{min}.

\( \alpha - \beta \) pruning example.
The α–β algorithm function \( \text{Alpha-Beta-Decision}(\text{state}) \) returns an action that maximizes \( \text{Max-Value}(\text{state}, \alpha, \beta) \).

The \( \text{Max-Value}(\text{state}, \alpha, \beta) \) function returns a utility value given state, \( \alpha \), and \( \beta \).

Inputs:
- \( \text{state} \), current state in game
- \( \alpha \), the value of the best alternative for max along the path to state
- \( \beta \), the value of the best alternative for min along the path to state

If \( \text{Terminal-Test}(\text{state}) \) then return \( \text{Utility}(\text{state}) \), which is a terminal state.

\[
v \leftarrow -\infty
\]
for \( a, s \) in \( \text{Successors}(\text{state}) \) do:

\[
v \leftarrow \text{Max}(v, \text{Min-Value}(s, \alpha, \beta))
\]
if \( v \geq \beta \) then return \( v \)

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

The \( \text{Min-Value}(\text{state}, \alpha, \beta) \) function returns a utility value similar to \( \text{Max-Value} \) but with roles of \( \alpha \) and \( \beta \) reversed.

**Properties of α–β Pruning**

Pruning does not affect final result.
Good move ordering improves effectiveness of pruning.
With "perfect ordering," time complexity is \( O(b^{m/2}) \), which 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!

**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.*

**Evaluation functions**

Black to move
White slightly better
White to move
Black winning

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**

Behavior is preserved under any monotonic transformation of Eval

\[
\max \text{Behavior} \left( O(V) \right) = \max \text{Behavior} \left( 20 \cdot \text{Eval} \right)
\]

**Resource limits**

Standard approach:
- Use \( \text{Cutoff-Test} \) instead of \( \text{Terminal-Test} \).
- Use \( \text{Eval} \) instead of \( \text{Utility} \).

E.g., evaluation function that estimates desirability of position.

Suppose we have 100 seconds.

\[
\text{nodes/second} \approx \frac{20}{20} \approx 35\text{,}800
\]

⇒ \( \alpha–\beta \) reaches depth 8 ⇒ pretty good chess program.
Nondeterministic games in general

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

**Simplified example with coin-flipping:**

```plaintext
MIN

2

MAX

4 7 4 6 0 5 −2

0.5 0.5 0.5 0.5

3 −1
```

Algorithm for nondeterministic games

*Expectiminimax* gives perfect play

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

1. If *state* is a *Max* node
   - return the highest *Expectiminimax*-value of *Successors*(*state*)
2. If *state* is a *Min* node
   - return the lowest *Expectiminimax*-value of *Successors*(*state*)
3. If *state* is a chance node
   - return average of *Expectiminimax*-value of *Successors*(*state*)

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

**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.

```
DICE

MIN

MAX

2 2 3 3 1 1 4 4

2 3 1 4

.9 .1 .9 .1

2.1 1.3
```

**TDGammon** uses depth-2 search + very good *Eval* ≈ world-champion level

Games of imperfect information in general

```
Non-deterministic games: backgammon
```

```plaintext
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

Chapter 6 26

Algorithm for nondeterministic games

*Expectiminimax* gives perfect play

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

1. If *state* is a *Max* node
   - return the highest *Expectiminimax*-value of *Successors*(*state*)
2. If *state* is a *Min* node
   - return the lowest *Expectiminimax*-value of *Successors*(*state*)
3. If *state* is a chance node
   - return average of *Expectiminimax*-value of *Successors*(*state*)

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

**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.

```
DICE

MIN

MAX

2 2 3 3 1 1 4 4

2 3 1 4

.9 .1 .9 .1

2.1 1.3
```

**TDGammon** uses depth-2 search + very good *Eval* ≈ world-champion level

Games of imperfect information in general

```
Non-deterministic games: backgammon
```
**Example**

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

<table>
<thead>
<tr>
<th>8</th>
<th>9</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4</th>
<th>2</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

---

**Example**

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

<table>
<thead>
<tr>
<th>8</th>
<th>9</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4</th>
<th>2</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

-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.

-0.5

**Example**

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

<table>
<thead>
<tr>
<th>8</th>
<th>9</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4</th>
<th>2</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

**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.

-0.5

**Example**

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

<table>
<thead>
<tr>
<th>8</th>
<th>9</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4</th>
<th>2</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

-0.5

**Example**

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

<table>
<thead>
<tr>
<th>8</th>
<th>9</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4</th>
<th>2</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

0
Proper analysis of all random actions is impossible. Actions depend on information state, not real state. Uncertainty constrains the assignment of values to states.

Good idea to think about what to think about. Partially observable must approximate. They illustrate several important points about all games and are fun to work on (and dangerous).

| Summary |

Games are fun to work on! (and dangerous)

- They illustrate several important points about all games and are fun to work on (and dangerous).
- Games are to AI as grand prix racing is to automobile design.
- They are fun to work on (and dangerous).
- They illustrate several important points about all games and are fun to work on (and dangerous).
- Games are fun to work on (and dangerous).

**Proper analysis**

The value of an action is the average of its values in all actual states. With partial observability, the value of an action depends on the information state, not the real state.

1. Acting randomly to minimize information disclosure.
2. Signaling to one's partner.
3. Acting to obtain information.

This leads to rational behaviors such as:

- Acting randomly to minimize information disclosure.
- Signaling to one's partner.
- Acting to obtain information.

In all actual states, the agent is in an information state, the value of an action depends on the information state, not the real state. With partial observability, the value of an action depends on the information state.