Adversarial Search
by Stuart Russell
modified by Jacek Malec for LTH lectures
January 24th, 2018

Chapter 5 of AIMA

Outline

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

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, 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>chess, checkers, go, othello</td>
<td>backgammon monopoly</td>
</tr>
<tr>
<td>battleships, blind tictactoe</td>
<td>bridge, poker, scrabble nuclear war</td>
</tr>
</tbody>
</table>

\( \copyright \) Stuart Russell

Chapter 5 of AIMA
**Game tree (2-player, deterministic, turns)**

**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:

$$\text{MAX}$$

**Minimax algorithm**

```python
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)
    $v \leftarrow -\infty$
    for $a, s$ in Successors(state) do $v \leftarrow \max(v, \text{Min-Value}(s))$
    return $v$

function Min-Value(state) returns a utility value
    if Terminal-Test(state) then return Utility(state)
    $v \leftarrow \infty$
    for $a, s$ in Successors(state) do $v \leftarrow \min(v, \text{Max-Value}(s))$
    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??

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

\[
\begin{array}{c}
\text{MAX} \\
\text{MIN} \\
3 & 12 & 8
\end{array}
\]

\(\alpha-\beta\) pruning example

\[
\begin{array}{c}
\text{MAX} \\
\text{MIN} \\
3 & 12 & 8 & 2
\end{array}
\]

\(\alpha-\beta\) pruning example

\[
\begin{array}{c}
\text{MAX} \\
\text{MIN} \\
3 & 12 & 8 & 2 & 14
\end{array}
\]

\(\alpha-\beta\) pruning example

\[
\begin{array}{c}
\text{MAX} \\
\text{MIN} \\
3 & 12 & 8 & 2 & 14 & 5
\end{array}
\]
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
Use additional heuristics (e.g. Killer Moves)
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

$$\Rightarrow 10^6 \text{ nodes per move } \approx 33^{7.5}$$

$$\Rightarrow \alpha-\beta \text{ reaches depth 8 } \Rightarrow \text{pretty good chess program}$$

### 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

$$f_1(s) = \text{(number of white queens)} - \text{(number of black queens)}, \text{ etc.}$$

### 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.

**Chess:** Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.

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

**Go:** In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves. AlphaGo defeated Lee Sedol, currently the best human go player, in March 2016. AlphaGo uses Monte Carlo tree search, guided by evaluation functions learnt by deep NNs.
Selection

Expansion

Simulation

Backpropagation

Monte Carlo Tree Search

© Stuart Russell
Chapter 5 of AIMA
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 then
   return the highest \( \text{ExpectiMinimax-Value of Successors}(\text{state}) \)
2. If state is a MIN node then
   return the lowest \( \text{ExpectiMinimax-Value of Successors}(\text{state}) \)
3. If state is a chance node then
   return average of \( \text{ExpectiMinimax-Value of Successors}(\text{state}) \)

Nondeterministic games in practice

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

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

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 \text{Eval}
\( \approx \) 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. *

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

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)

◊ perfection is unattainable ⇒ must approximate
◊ good idea to think about what to think about
◊ uncertainty constrains the assignment of values to states
◊ optimal decisions depend on information state, not real state

Games are to AI as grand prix racing is to automobile design