Adversarial Search

Chapter 5

Sections 1-5
Outline

• Games
• Optimal decisions
• $\alpha$-$\beta$ pruning
• Imperfect, real-time decisions
• Stochastic Games
Game Playing State-of-the-Art

- **Checkers**: Chinook ended a 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. Checkers is now solved!

- **Chess**: Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997. Deep Blue examined 200 million positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.

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

- **Go**: Human champions are beginning to be challenged by machines, though the best humans still beat the best machines. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves, along with aggressive pruning.

- **Pacman**: unknown

- **AAAI conferences** now have *general* game-playing competitions, with a $10K$ prize!
Game Search

• Game-playing programs developed by AI researchers since the beginning of the modern AI era (chess, checkers in 1950s)

• Game Search
  – Sequences of player’s decisions we control
  – Decision of other player(s) we do not control

• Contingency problem: many possible opponent’s moves must be “covered” by the solution
  – Introduces uncertainty to the game since we do not know what the opponent will do

• Rational opponent: maximizes it’s own utility function
Types of Game Problems

- **Adversarial**
  - Win of one player is a loss of the other
  - Focus of this course

- **Cooperative**
  - Players have common interests and utility function

- A spectrum of others in between
Typical AI “Games”:

- Deterministic and Fully Observable Environment
- Two agents with turn-taking for actions
- Zero-sum (adverserial)
- Abstract (robotic soccer notable exception)
  - state easy to represent, few action choices, well-defined goals
  - hard to solve
<table>
<thead>
<tr>
<th>Perfect Information</th>
<th>Deterministic</th>
<th>Chance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tic Tac Toe, Chess</td>
<td></td>
<td>Backgammon</td>
</tr>
<tr>
<td>Imperfect information</td>
<td>Stratego</td>
<td>Poker, Bridge</td>
</tr>
</tbody>
</table>
Deterministic Single-Player

- Deterministic, single player, perfect information:
  - Know the rules
  - Know what actions do
  - Know when you win
  - E.g. Freecell, 8-Puzzle, Rubik’s cube

- … it’s just search!

- Slight reinterpretation:
  - Each node stores a value: the best outcome it can reach
  - This is the maximal outcome of its children (the max value)
  - Note that we don’t have path sums as before (utilities at end)

- After search, can pick move that leads to best node
Deterministic Two-Player

• E.g. tic-tac-toe, chess, checkers
• Zero-sum games
  – One player maximizes result
  – The other minimizes result
• **Minimax search**
  – A state-space search tree
  – Players alternate
  – Choose move to position with highest minimax value = best achievable utility against best play
Game Search

• Problem Formulation
  – **Initial state**: initial board position + information about whose move it is
  – **Successors**: legal moves a player can make
  – **Goal (terminal test)**: determines when the game is over
  – **Utility function**: measures the outcome of the game and its desirability

• Search objective
  – Find the sequence of player’s decisions (moves) maximizing its utility
  – Consider the opponent’s moves and their utility
Game Tree

• Initial State and Legal Moves for Each Side
Game Tree
(2-player, deterministic, turns)
Game Tree
(2-player, deterministic, turns)

MAX (X)

MIN (O)

MAX (X)

MIN (O)

TERMINAL

Utility

-1 0 +1

- MAX and MIN are the 2 players
- MAX goes first
- Players then take turns
Game Tree
(2-player, deterministic, turns)

• MAX has 9 possible legal first moves (ignoring symmetry)
Game Tree
(2-player, deterministic, turns)

- Utility of terminal states (when game is over) is from MAX’s point of view
- Points are awarded to both players at the end of the game
  - -1 is a loss
  - 0 is a draw
  - 1 is a win
Minimax Algorithm

• How do we deal with the contingency problem?

  – Assuming that the opponent is rational and always optimizes its behavior (opposite to us), we consider the opponent’s best response
  – Then the minimax algorithm determines the best move
Minimax

- Finds an optimal (contingent) strategy, assuming perfect play for deterministic games

- Idea: choose move to position with highest MINIMAX VALUE = best achievable payoff against best play

- MINIMAX-VALUE (n)
  - UTILITY (n) if n is a terminal state
  - max_s MINIMAX-VALUE (s) if n is a MAX node
  - min_s MINIMAX-VALUE (s) if n is a MIN node

  (where s is an element of the successors of n)
Minimax Example
Properties of minimax

- **Complete?** Yes (if tree is finite)
- **Optimal?** Yes (against an optimal opponent)
- **Time complexity?** $O(b^m)$
- **Space complexity?** $O(bm)$ (depth-first exploration)

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

- Do we really need to explore every path???
Solutions to the Complexity Problem

• Dynamic pruning of redundant branches of the search tree
  – Some branches will never be played by rational players since they include sub-optimal decisions (for either player)
    • Identify a provably suboptimal branch of the search tree before it is fully explored
    • Eliminate the suboptimal branch
  – Procedure: Alpha-Beta Pruning

• Early cutoff of the search tree
  – Use imperfect minimax value estimate of non-terminal states
α-β pruning example

```
MAX

MIN

3
12
8

≥ 3
```
α-β pruning example
α-β pruning example
α-β pruning example
α-β pruning example
\( \alpha-\beta \) pruning example

\[
\text{MINIMAX-VALUE(root)} = \max(\min(3, 12, 8), \min(2, x, y), \min(14, 5, 2))
\]
\[
= \max(3, \min(2, x, y), 2)
\]
\[
= \max(3, z, 2) \text{ for } z \leq 2
\]
\[
= 3
\]
Properties of α-β

• Pruning does not affect final result

• Good move ordering improves effectiveness of pruning

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

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

Recap

– Minimax explores the full search space
– Alpha Beta prunes, but still searches all the way to terminal states for a portion of the search space

Standard approaches to fix resource limits

– **cutoff test:**
  e.g., depth limit

– **evaluation function**
  = 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
  \[ f_1(s) = (\text{number of white queens}) - (\text{number of black queens}), \text{ etc.} \]
Cutting off search

MinimaxCutoff is identical to MinimaxValue except

1. Terminal? is replaced by Cutoff?
2. Utility is replaced by Eval

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
Stochastic Single-Player

• What if we don’t know what the result of an action will be? E.g.,
  – In solitaire, shuffle is unknown
  – In minesweeper, mine locations
  – In pacman, ghosts!

• Can do expectimax search
  – Chance nodes, like actions except the environment controls the action chosen
  – Calculate utility for each node
  – Max nodes as in search
  – Chance nodes take average (expectation) of value of children
Stochastic Two-Player

- E.g. backgammon
- Expectiminimax (!)
  - Environment is an extra player that moves after each agent
  - Chance nodes take expectations, otherwise like minimax

\[
\text{if } state \text{ is a Max node then } \quad \text{return the highest } \text{ExpectiMinimax-Value of Successors}(state) \\
\text{if } state \text{ is a Min node then } \quad \text{return the lowest } \text{ExpectiMinimax-Value of Successors}(state) \\
\text{if } state \text{ is a chance node then } \quad \text{return average of } \text{ExpectiMinimax-Value of Successors}(state)
\]
Minimax for nondeterministic games

```
  MAX

    3

      0.5

        2

        4

        7

        4

    0.5

      0

        6

        0

        5

        -2

MIN

-1

  CHANCE

```
Stochastic Two-Player

- Dice rolls increase $b$: 21 possible rolls with 2 dice
  - Backgammon $\approx 20$ legal moves
- As depth increases, probability of reaching a given node shrinks
  - So value of lookahead is diminished
  - So limiting depth is less damaging
  - But pruning is less possible…
- TDGammon uses depth-2 search + very good eval function + reinforcement learning: world-champion level play
Summary

• Games are fun to work on!
• They illustrate several important points about AI
• perfection is unattainable → must approximate
• good idea to think about what to think about