Game-playing AIs: Games and Adversarial Search I

AIMA 5.1-5.2

Games: Outline of Unit

Part I: Games as Search
- Motivation
- Game-playing AI successes
- Game Trees
- Evaluation Functions

Part II: Adversarial Search
- The Minimax Rule
- Alpha-Beta Pruning

May 11, 1997

Ratings of human & computer chess champions

(from: The Guardian)

AlphaGo beats Ke Jie again to wrap up three-part match

The Simplest Game Environment

- **Multiagent**
- **Static:** No change while an agent is deliberating
- **Discrete:** A finite set of percepts and actions
- **Fully observable:** An agent's sensors give it the complete state of the environment.
- **Strategic:** The next state is determined by the current state and the action executed by the agent and the actions of one other agent.

Key properties of our sample games

1. Two players alternate moves
2. Zero-sum: one player's loss is another's gain
3. Clear set of legal moves
4. Well-defined outcomes (e.g. win, lose, draw)

- **Examples:**
  - Chess, Checkers, Go,
  - Mancala, Tic-Tac-Toe, Othello,
  - Nim, …

More complicated games

- Most card games (e.g. Hearts, Bridge, etc.) and Scrabble
  - Stochastic, not deterministic
  - Not fully observable: lacking in perfect information
- Real-time strategy games (lack alternating moves). e.g. Warcraft
- Cooperative games

A *cooperative multi-agent environment*: Pragbot

How to Play a Game by Searching

- **General Scheme**
  1. Consider all legal successors to the current state ('board position')
  2. Evaluate each successor board position
  3. Pick the move which leads to the best board position.
  4. After your opponent moves, repeat.

- **Design issues**
  1. Representing the ‘board’
  2. Representing legal next boards
  3. Evaluating positions
  4. Looking ahead

Formalizing the Game setup

1. Two players: **MAX** and **MIN**: **MAX** moves first.
2. **MAX** and **MIN** take turns until the game is over.
3. Winner gets award, loser gets penalty.

- **Games as search:**
  - **Initial state:** e.g. board configuration of chess
  - **Successor function:** list of (move, state) pairs specifying legal moves.
  - **Terminal test:** Is the game finished?
  - **Utility function:** Gives numerical value of terminal states. e.g. win (+∞), lose (−∞) and draw (0)
  - **MAX** uses search tree to determine next move.
**Hexapawn: A very simple Game**

- Hexapawn is played on a 3x3 chessboard

- Only standard pawn moves:
  1. A pawn moves forward one square onto an empty square
  2. A pawn "captures" an opponent pawn by moving diagonally forward one square, if that square contains an opposing pawn. The opposing pawn is removed from the board.

**Hexapawn: Three Possible First Moves**

- Player $P_1$ wins the game against $P_2$ when:
  - One of $P_1$'s pawns reaches the far side of the board.
  - $P_2$ cannot move because no legal move is possible.
  - $P_2$ has no pawns left.

  (Invented by Martin Gardner in 1962, with learning "program" using match boxes. Reprinted in "The Unexpected Hanging..."

**Game Trees**

- Represent the game problem space by a tree:
  - Nodes represent 'board positions'; edges represent legal moves.
  - Root node is the first position in which a decision must be made.

**MAX & MIN Nodes : An egocentric view**

- Two players: MAX, MAX's opponent MIN
- All play is computed from MAX's vantage point.
- When MAX moves, MAX attempts to MAXimize MAX's outcome.
- When MAX's opponent moves, they attempt to MINimize MAX's outcome.

  WE TYPICALLY ASSUME MAX MOVES FIRST:

- Label the root (level 0) MAX
- Alternate MAX/MIN labels at each successive tree level (ply).
- Even levels represent turns for MAX
- Odd levels represent turns for MIN

MAX (von Sydow) plays chess...
**Game Trees**

- Represent the game problem space by a tree:
  - Nodes represent 'board positions', edges represent legal moves.
  - Root node is the first position in which a decision must be made.
- Evaluation function $f$ assigns real-number scores to 'board positions' without reference to path
- Terminal nodes represent ways the game could end, labeled with the desirability of that ending (e.g. win/lose/draw or a numerical score)

**Evaluation functions:** $f(n)$

- Evaluates how good a 'board position' is
- Based on *static features* of that board alone
- Zero-sum assumption lets us use one function to describe goodness for both players.
  - $f(n)>0$ if MAX is winning in position $n$
  - $f(n)=0$ if position $n$ is tied
  - $f(n)<0$ if MIN is winning in position $n$
- Build using expert knowledge,
  - Tic-tac-toe: $f(n)=(\# \text{ of } 3 \text{ lengths open for MAX})-(\# \text{ open for MIN})$

**Chess Evaluation Functions**

- Alan Turing's $f(n)=(\text{sum of } A\text{'s piece values})-(\text{sum of } B\text{'s piece values})$

- More complex: weighted sum of positional features:
  - Deep Blue has $>8000$ features

  - Pawn $1.0$
  - Knight $3.0$
  - Bishop $3.25$
  - Rook $5.0$
  - Queen $9.0$

  Positive: rooks on open files, knights in closed positions, control of the center, developed pieces

  Negative: doubled pawns, wrong-colored bishops in closed positions, isolated pawns, pinned pieces

**Some Chess Positions and their Evaluations**

- White to move $f(n)=(9+3)-(5+5+3.25) = -1.25$

- ... Nxg5?? $f(n)=(9+3)-(5+5) = -2$

- Uh-oh: $f(n)=(3)-(5+5) = -7$

So, considering our opponent's possible responses would be wise.

**The Minimax Rule (AIMA 5.2)**
The Minimax Rule: 'Don’t play hope chess'

Idea: Make the best move for MAX assuming that MIN always replies with the best move for MIN

Easily computed by a recursive process
- The backed-up value of each node in the tree is determined by the values of its children:
  - For a MAX node, the backed-up value is the maximum of the values of its children (i.e. the best for MAX)
  - For a MIN node, the backed-up value is the minimum of the values of its children (i.e. the best for MIN)

The Minimax Procedure

Until game is over:
1. Start with the current position as a MAX node.
2. Expand the game tree a fixed number of ply.
3. Apply the evaluation function to the leaf positions.
4. Calculate back-up values bottom-up.
5. Pick the move assigned to MAX at the root
6. Wait for MIN to respond

2-ply Example: Backing up values

What if MIN does not play optimally?
- Definition of optimal play for MAX assumes MIN plays optimally:
  - Maximizes worst-case outcome for MAX.
  - (Classic game theoretic strategy)
- But if MIN does not play optimally, MAX will do even better. [Theorem-not hard to prove]

Comments on Minimax Search
- Depth-first search with fixed number of ply m as the limit.
  - $O(b^m)$ time complexity – As usual
  - $O(m)$ space complexity
- Performance will depend on
  - the quality of the static evaluation function (expert knowledge)
  - depth of search (computing power and search algorithm)
- Differences from normal state space search
  - Locking to make one move only, despite deeper search
  - No cost on arcs – costs from backed-up static evaluation
  - MAX can’t be sure how MIN will respond to his moves
- Minimax forms the basis for other game tree search algorithms.

IF TIME ALLOWS....
Alpha-Beta Pruning (AIMA 5.3)

Many slides adapted from Richard Lathrop, USC/ISI, CS 271

Alpha-Beta Pruning

• A way to improve the performance of the Minimax Procedure
• Basic idea: “If you have an idea which is surely bad, don’t take the time to see how truly awful it is” – Pat Winston

\[ \alpha \geq \text{MIN's current upper bound on MIN's outcome} \]
\[ \beta \geq \text{MAX's current lower bound on MAX's outcome} \]

Therefore, stop evaluating a branch whenever:
• When evaluating a MAX node: a value \( v \geq \beta \) is backed up
  — MIN will never select that MAX node
• When evaluating a MIN node: a value \( v \leq \alpha \) is found
  — MAX will never select that MIN node

Alpha-Beta Pruning II

• During Minimax, keep track of two additional values:
  • \( \alpha \): MAX's current lower bound on MAX's outcome
  • \( \beta \): MIN's current upper bound on MIN's outcome
• MAX will never allow a move that could lead to a worse score (for MAX) than \( \alpha \)
• MIN will never allow a move that could lead to a better score (for MAX) than \( \beta \)

Alpha-Beta Pruning Illa

• Based on observation that for all viable paths utility value \( f(n) \) will be \( \alpha \leq f(n) \leq \beta \)
• Initially, \( \alpha = -\infty \), \( \beta = \infty \)
• As the search tree is traversed, the possible utility value window shrinks as \( \alpha \) increases, \( \beta \) decreases

Alpha-Beta Pruning Illib

• Whenever the current ranges of alpha and beta no longer overlap, it is clear that the current node is a dead end

Tic-Tac-Toe Example with Alpha-Beta Pruning

Figure 4.17: Two-ply minimax applied to the opening move of tic-tac-toe.