Artificial Intelligence

Minimax and Alpha-Beta Pruning

In which we examine the problems that arise when we try to plan ahead in a world that includes a hostile agent (other agent planning against us).
Games

- Adversarial search problems and Game theory
  - Competitive environments in which goals of multiple agents are in conflict (often known as games)
- Game playing is idealization of worlds in which hostile agents act so as to diminish one’s well-being!
  - Games problems are like real world problems 😊
- Classic AI games
  - Deterministic, turn-taking, two-player, perfect information

Classic AI Games

- State of game easy to represent
- Agents usually restricted to fairly small number of well-defined actions
- NEW: **Opponent introduces uncertainty**
- Games usually too hard to solve directly
  - Chess
    - Branching factor 35
    - Often go to 50 moves by each player
    - About $35^{100}$ nodes!
- Therefore, games are a good domain to study
  - *But is this really intelligence?!*
AI Game Play

- Define optimal move and need algorithm for finding it
- Ignore portions of search tree that make no difference to final choice
  - “Pruning”

A Game Defined as Search Problem

- **Initial state**
  - Board position
  - Whose move it is
- **Operators** (successor function)
  - Defines legal moves and resulting states
- **Terminal (goal) test**
  - Determines when game is over (terminal states)
- **Utility (objective, payoff) function**
  - Gives numeric value for the game outcome at terminal states
  - e.g., \{win = +$1, loss = -$1, draw = 0\}
Optimal Strategies:
Perfect Decisions in Two-Person Games

- Two players
  - MAX (you)
  - MIN
- Turn-taking: MAX moves first, then take turns with MIN moving until game over
- At end, points awarded to winning player
  - Or penalties given to loser
- Can formulate this gaming structure into a search problem
An Opponent

- If were normal search problem, then MAX (you/agent) need only search for sequence of moves leading to winning state
- But, MIN (the opponent) has input
- MAX must use a “strategy” that will lead to a winning state, regardless of what MIN does
  - Strategy picks best move for MAX in relation to all possible moves by MIN
Strategy and Techniques

• “Minimax”
  – Determines the best moves for MAX, assuming that MAX and opponent (MIN) play perfectly
  – Decides best opening first move for MAX
  – Serves as basis for analysis of games and algorithms

• Alpha-beta pruning
  – Ignore portions of search tree that make no difference to final choice

Playing Perfectly?

[The game hasn't yet started]

**FRATBOT #2:**
“Mate in 143 moves.”

**FRATBOT #3:**
“Oh, poo, you win again!”
Minimax

- Requires perfect play for two player (MAX, MIN) deterministic games
- Task: Choose (and move to) position with highest minimax value
  - Best achievable payoff against best play
  - Maximizes the worst-case outcome for MAX

Minimax Algorithm

- Generate whole game tree (or from current state downward – depth-first process online)
  - Initial state(s) to terminal states
- Apply utility function to terminal states
  - Get payoff for the different final moves of game
- Use utilities at terminal states to determine utility of nodes one level higher in tree
  - e.g., Find MIN’s best attempt to minimize high payoff for MAX at terminal level
- Continue backing up the values to the root
  - One layer at a time
- Value at root determines the best payoff and opening move for MAX (minimax decision)
2-Ply Minimax Game
(one move for each player)
Properties of Minimax

• Complete
  – If tree is finite

• Time
  – Depth-first exploration
  – \(O(b^m)\), max depth of \(m\) with \(b\) legal moves at each point
    (impractical for real games)

• Space
  – Depth-first exploration
  – \(O(bm)\)

• Optimality
  – Yes against an optimal opponent
    • Does even better when MIN not play optimally (making mistakes)

“Pruning”

• Minimax search has to search large number of states
• But possible to compute correct minimax decision without looking at every node in search tree
• Eliminating a branch of search tree from consideration (without looking at it) is called “pruning”
• Alpha-beta pruning
  – Prunes away branches that cannot possibly influence final minimax decision
  – Returns same move as general minimax
Alpha-Beta Pruning

- Can be applied to trees of any depth
- Often possible to prune entire subtrees rather than just leaves
- Alpha-beta name
  - Alpha = value of best (highest-value) choice found so far at any choice point along path for MAX
  - Beta = value of best (lowest-value) choice found so far at any choice point along path for MIN
Alpha-Beta Pruning

MAX

MIN

MAX

\[ m \geq m \leq n \]

If \( n \) is worse than \( m \), MAX will prune.

\( m \) is best value (to MAX) so far on current path.

"If \( n \) is worse than \( m \)", MAX will prune.
Alpha-Beta Pruning

MAX

MIN

Terminal

\[ \geq 3 \]

\[ 3 \]

\[ 12 \]

\[ 8 \]
Alpha-Beta Pruning

MAX

MIN

Terminal

α ≥ 3

β ≤ 2

α

Terminal

α

Terminal

α ≥ 3

β ≤ 2

β ≤ 14

α
Alpha-Beta Pruning

MAX

MIN

Terminal

Alpha-Beta Pruning

MAX

MIN

Terminal
In-Class Exercise

Node Ordering

- Good move ordering would improve effectiveness of pruning
  - Prunes faster
    - e.g., want to have children with values ordered as 1, 10, 100 (not 100, 10, 1)

This ordering has a better chance of pruning the 10 and 100.
Properties of Alpha-Beta

• Pruning does not affect final result
• With “perfect ordering” the time complexity is greatly reduced
• A simple example of the value of “reasoning about which computations are relevant”
  – Meta-reasoning (reasoning about reasoning)

Games with Chance

• Many games have a random element
  – e.g., throwing dice to determine next move
• Cannot construct standard game tree as before
  – As in Tic-Tac-Toe
• Need to include “CHANCE nodes”
• Branches leading from chance node represent the possible chance-outcomes and probability
  – e.g., die rolls: each branch has the roll value (1-6) and its chance of occurring (1/6th)
**ExpectiMiniMax**

- TERMINAL, MAX, MIN nodes work same way as before
- CHANCE nodes are evaluated by taking **weighted average** of values resulting from all possible chance outcomes (e.g., die rolls)
- Process is backed-up recursively all the way to root (as before)

**Simple Example**

Move $A_1$ is “expected” to be best for MAX

NOTE: Alpha-Beta can be adapted to prune with chance nodes
Early Game Programs

• Chess
  – Most attention
  – In 1957, predicted computer would beat world champion in 10 years (off by 40 years)
  – Deep Blue defeated Garry Kasparov (6 game match)

“The decisive game of the match was Game 2, which left a scar in my memory … we saw something that went well beyond our wildest expectations of how well a computer would be able to foresee the long-term positional consequences of its decisions. **The machine refused to move to a position that had a decisive short-term advantage – showing a very human sense of danger**”
  – (Kasparov, 1997)

Early Game Programs

• Chess (con’t)
  – Searched 126 million nodes per second on average
    • Peak speed of 330 million nodes per second
  – Generated up to 30 billion positions per move
    • Reaching depth 14 routinely
  – Heart of machine was iterative-deepening alpha-beta search
    • Also generated extensions beyond depth limit for sufficiently interesting lines of moves
  – Later Deep Fritz ended in draw in 2002 against world champion Vladimir Kramnik
    • Ran on ordinary PC (not a supercomputer)

• Go
  – Branching factor of 361! (chess is 35)
    • Regular search methods no good
  – AlphaGo more recently…
Summary

• Games can be defined as search problems
  – With complexity of real world problems
• Minimax algorithm determines the best move for a player
  – Assuming the opponent plays perfectly
  – Enumerates entire game tree
• Alpha-beta algorithm similar to minimax, but prunes away branches that are irrelevant to the final outcome
  – May need to cut off search at some point if too deep
• Can incorporate “chance”