Material

• Reading: Read chapters 6
• Read chapter 7 (for Monday).
  – expect new homework on Monday (this will be a long one).
• Today’s lecture covers chapter 6.
• Announcements: Still working on makeup class times. Send me email if you haven’t already for Friday schedules.

Outline

• Optimal decisions
• $\alpha$-$\beta$ pruning
• Imperfect, real-time decisions

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
• History:
  – Computer considers possible lines of play (Babbage, 1846)
  – Algorithm for perfect play (Zermelo, 1912; Von Neumann, 1944)
  – Finite horizon, approximate evaluation (Zuse, 1945; Wiener, 1946; 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

| deterministic | chess, checkers, go, checkers |
| imperfect information | battleships, bridge, poker, scrabble |
| perfect information | monopoly |
| chance | nuclear war |
Game tree (2-player, deterministic, turns)

Minimax
- Perfect play for deterministic games
- Idea: choose move to position with highest minimax value
  = best achievable payoff against best play
- E.g., 2-ply game (ply == move by one player):

Minimax algorithm

Properties of minimax
- Complete? Yes, if tree is finite. Finite strategy can exist even if tree is infinite.
- Optimal? Yes (against an optimal opponent).
  - for sub-optimal opponent, better strategies might exist.
- Time complexity? $O(b^m)$
- Space complexity? $O(b^m)$ (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
Why is it called \(\alpha-\beta\)?

- \(\alpha\) is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for \(\text{max}\).
- If \(v\) is worse than \(\alpha\), \(\text{max}\) will avoid it → prune that branch.
- Define \(\beta\) similarly for \(\text{min}\).

The \(\alpha-\beta\) algorithm

\[
\text{MAX-VALUE}(\text{state}) \quad \text{return} \quad \alpha \quad \text{as utility value}
\]

\[
\text{Inputs:} \text{state, current state in game} \\
\alpha, \text{the value of the best alternative for MAX along the path to state} \\
\beta, \text{the value of the best alternative for MIN along the path to state} \\
\text{if TERMINAL-TEST(state) then return UTILITY(state)} \\
\text{for } a, v \in \text{SUCCESSOR(state)} \\
\text{if } v \geq \alpha \text{ then return } v \\
\alpha \leftarrow \text{MAX}(a, \alpha) \\
\text{return } \alpha
\]

\[
\text{MIN-VALUE}(\text{state}, \alpha, \beta) \quad \text{return} \quad v \quad \text{as utility value} \\
\text{Inputs: state, current state in game} \\
\alpha, \text{the value of the best alternative for MAX along the path to state} \\
\beta, \text{the value of the best alternative for MIN along the path to state} \\
\text{if TERMINAL-TEST(state) then return UTILITY(state)} \\
\text{for } a, v \in \text{SUCCESSOR(state)} \\
\text{if } v \geq \alpha \text{ then return } v \\
\beta \leftarrow \text{MIN}(v, \text{MIN-VALUE}(a, \alpha, \beta)) \\
\text{return } v
\]
Properties of α-β

- Pruning does not affect final result
- Good move ordering improves effectiveness of pruning
- With "perfect ordering," time complexity = \( O(b^{m/2}) \)
  \( \rightarrow \) doubles the solvable tractable depth of search
- A simple example of the value of reasoning about which computations are relevant (a form of metareasoning)
- For chess, 35\(^6\) is still impossible!

Resource limits

Standard approach:

- cutoff test: use a cutoff-test rather than a terminal-test.
  e.g., depth limit (perhaps add quiescence search)
  quiescence == stop when nothing is going to make drastic changes
  (e.g., taking a queen)
- evaluation function: use Eval instead of Utility
  = eval estimates desirability of position (but is a heuristic, since the exact true evaluation requires a full search)
Suppose we have 100 secs to move, can explore \( 10^6 \) nodes/sec
  \( \approx \) 35\(^6\) = \( \alpha \)-\( \beta \) reaches depth 8, relatively good chess program

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

Exact values don’t matter

Behavior is preserved under any monotonic transformation
of Eval function.
Only the order matters: payoff in deterministic games acts
as an ordinal utility function (in game theory)

Cutting off search

\textit{MinimaxCutoff} is identical to \textit{MinimaxValue} except

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

Does it work in practice?

\( b^m = 10^6, b=35 \rightarrow m=4 \)
4-ply lookahead is a hopeless chess player!
- 4-ply = human novice
- 8-ply = typical PC, human master
- 12-ply = Deep Blue, Kasparov

Deterministic games in practice

- Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used a precomputed endgame database
  defining perfect play for all positions involving 8 or fewer pieces on
  the board, a total of 444 billion positions.
- Chess: Deep Blue defeated human world champion Garry 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: 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.
Non-Deterministic games: backgammon

Non-Deterministic games in general

- In nondeterministic games, chance is introduced by dice, card-shuffling, or coin flipping. In these cases, we have search nodes for “chance” which can be seen as a random player in the game. In the coin case:

Algorithm for non-deterministic games

- Expectedminimax gives perfect play
- Just like Minimax, except we also handle chance nodes.

... if state is a MAX node then
    return the highest \( \text{Expect}\text{Minimax-Value of Successors(state)} \)
... if state is a MIN node then
    return the lowest \( \text{Expect}\text{Minimax-Value of Successors(state)} \)
... if state is a chance node then
    return average of \( \text{Expect}\text{Minimax-Value of Successors(state)} \)

non-deterministic games in practice

- Dice rolls increase b: 21 possible rolls with 2 dice
- Backgammon \( \approx \) 20 legal moves (can be 6000 with 1-1 roll).
  - depth 4 = \( 20(21^2)20 \approx 1.2 \times 10^9 \)
- As depth increases, probability of reaching given node shrinks \( \Rightarrow \) value of lookahead is diminished
- \( \alpha-\beta \) pruning much less effective.
- TDGammon: uses depth-2 search + very good Eval (learned automatically), world champion level

Exact values DO matter.

Summary

- Games are fun to work on!
- They illustrate several important points about AI
- perfection is unattainable \( \Rightarrow \) must approximate
- good idea to think about what to think about