Artificial Intelligence

Introduction to game-playing search
Minimax algorithm, evaluation functions, alpha-beta pruning, advanced techniques

AI and computer game playing

- Game playing (especially chess and checkers) was the first test application of AI
- Search is performed to determine the best next move in the game
- The best move depends on what the opponent might do. As a result, this is sometimes called “adversary search”

Game tree search

- **Initial state**: initial board position and player
- **Operators**: one for each legal move
- **Terminal states**: a set of states that mark the end of the game
- **Utility function**: assigns numeric value to each terminal state
- **Game tree**: represents all possible game scenarios

Partial game tree for Tic-Tac-Toe

Minimax Algorithm

- Invented by Von Neumann & Morgenstern in 1944 as part of game theory
- Used for two player games.
- One player (MAX) is trying to maximize the value of the evaluation function.
- The opponent (MIN) is trying to minimize its value

Minimax Algorithm (cont.)

- Generate the game tree
- Apply the utility function to each terminal state to get its value
- “Back-up” values through game tree:
  - at each MAX node, take max of successors
  - at each MIN node, take min of successors
- Select best move for MAX node at root
Search complexity
- Space complexity of minimax using depth-first search is $O(bd)$ (where $b$ is branching factor and $d$ is depth of game tree). Why use depth-first search for minimax?
- Time complexity is $O(b^d)$
- For chess, $b \approx 35$, $d \approx 100$ for “reasonable” games, making exact solution infeasible

Static Evaluation Function
- Can’t search full game tree. Instead search to fixed depth or “ply” and use estimated values for minimax. (Idea due to Claude Shannon -- the inventor of information theory -- in 1950.)
- Static evaluation function estimates minimax values of non-terminal states. (Some claim evaluation function should measure $P$ (winning).)
- This strategy works because backed-up evaluations are better than static evaluations, and improve with the depth of “lookahead” tree.

Evaluating chess states
- “material” evaluation: count pieces for each side, giving each a weight (queen=9, rook=5, knight/bishop=3, pawn=1)
- Most evaluation functions are specified as a weighted sum of features. In chess, some features evaluate piece placement on the board and other features define configurations of several features. Deep Blue has about 6000 features in its evaluation function!
- The weights for chess features are usually learned using linear regression, hill-climbing, or a similar technique.
- The performance of a chess-playing program is very dependent on the quality of its evaluation function.

Alpha-Beta Pruning
- In chess, can only search full-width tree to about 4 levels
- The trick is to “prune” subtrees
- Fortunately, the best move is provably insensitive to certain subtrees
- If there exists a winning move at a node, then its sibling nodes (and their subtrees) need not be examined.
Alpha-Beta Pruning (cont.)

- Use bounds on minimax values to prune subtrees
- Bounds on MAX nodes are called **alpha values** and are initially \(-\infty\). The alpha value for a MAX node is the value of its best successor and can never decrease.
- Bounds on MIN nodes are called **beta values** and are initially \(\infty\). The beta value for a MIN node is the value of its worst successor and can never increase.
- Alpha cutoff: A MIN node can be pruned if its beta value is \(\leq\) the alpha value of its MAX parent
- Beta cutoff: A MAX node can be pruned if its alpha value \(\geq\) the beta value of its MIN parent.

Effectiveness of Alpha-Beta

- Alpha-beta pruning does not affect minimax value. It just improves efficiency of search.
- How much it improves efficiency of search depends on ordering of successors.
- With perfect ordering, can search twice as deep in given amount of time (i.e., *effective branching factor* is \(\sqrt{b}\)).
- Perfect ordering cannot be achieved, but simple ordering heuristics are very effective.

Node ordering

- Instead of exploring the children of a node in a fixed order (e.g., left to right), base the order of exploration on static evaluation of the nodes
- Expand children of MAX nodes in decreasing order of static evaluation
- Expand children of MIN nodes in increasing order of static evaluation
- This technique significantly improves performance of alpha-beta pruning
### Review questions

- Before moving on to some advanced topics, let’s review some of the basics
- What is maximum improvement possible with alpha-beta pruning and how is it possible to achieve it?
- Why is minimax performed using depth-first search instead of breadth-first search?
- With alpha-beta pruning, the effective branching factor for chess is about 6. For chess, what increase in computer speed is needed to search one ply deeper in the same amount of time? How much of the improvement in chess-playing programs is due to faster computers?

### Effectiveness of Alpha-Beta (cont.)

- Full-width minimax search in chess allows about 4-ply lookahead. With 4-ply lookahead, a chess program performs poorly and at the level of a human novice.
- For Deep Blue, alpha-beta pruning reduced the effective branching factor from about 35 to 6. This allows 8-ply lookahead. With 8-ply lookahead, a chess program performs at the level of a human master. But Deep Blue can actually perform 12-ply lookahead. Now, we review some additional techniques that it uses to make this possible.

### Quiescence and Secondary Search

- If a node represents a state in the middle of an exchange of pieces, the evaluation function may not give a reliable estimate of board quality. Example: after you capture a knight the evaluation may be good, but this is misleading if opponent is about to capture your queen.
- Solution: if node evaluation is not “quiescent,” continue alpha-beta search below that node but limit moves to those that significantly change evaluation function (e.g., capture moves, promotions). The branching factor for such moves is small.
- Deep Blue uses this technique to search some paths to depth 25, though it ordinarily only searches to depth 14.

### Iterative Deepening

- Given uncertainty about how deep a program can search in a given amount of time, repeat searches w/ maximum depth 1, 2, 3, etc., until time to think about move expires. Save best move found in previous search before starting new one.
- Is much search effort “wasted” by this approach?

### Transposition Tables

- Basic idea is caching: once position is evaluated, save in hash table to avoid re-evaluating.
- Called “transposition” tables because different orderings (transpositions) of same set of moves can lead to same position.
- Converts search tree to search graph (Chess game tree has approximately $35^{100}$ nodes while chess game graph has approximately $10^{40}$).
- Deep Blue: huge transposition tables (100,000,000+) must be carefully managed.

### Opening Books and Endgame Databases

- Centuries of chess experience have resulted in a set of initial moves that have been shown to be best. These are stored in a database for the chess program.
- Similar to openings, a wide collection of end positions has been accumulated and optimal strategies for each are known. (Computers have helped in determining these.) These are also stored in a database.
- Why is it easier to determine optimal play near the end of a game?
Special-Purpose Hardware and Parallel Processing

- Evaluation functions and move generators can be encoded in special-purpose chips to make them much faster. For example, 220 special-purpose VLSI chess chips were designed for Deep Blue.
- Deep Blue also used 32 general processors to conduct parallel search.

How deep does Deep Blue search?

- Hard to say when both alpha-beta pruning and secondary search are used:
  - minimum: six moves in typical middle-game positions
  - average: about eight moves
  - maximum: highly variable, but typically in the ten-to-twenty-move range

What is the difference between how a human plays chess and how Deep Blue plays chess?

- An experimenter showed chess positions to experts and non-players. Chess players were able to remember the chess positions accurately; non-players could not. When both groups were shown random chess boards, they did equally badly. This indicates chess experts recognize (and remember) meaningful patterns of the chess board. However, it’s difficult to design chess programs with the pattern-recognition abilities of humans (though some have tried).
- Instead, chess programs rely on what computers are good at – lots of calculation and search.

Is Deep Blue Intelligent?

Saying Deep Blue doesn’t really think about chess is like saying an airplane doesn’t really fly because it doesn’t flap its wings. — Drew McDermott

What do you think? Does the success of Deep Blue tell us anything about the nature of intelligence?