Teagan Clark

Dr. Forouraghi

Artificial Intelligence

6 November 2025

Assignment 4: The Game of Nim

Problem 1:

Game Description:

Nim is a turn-based game played using a set of given heaps. Once per turn, a player chooses one heap and removes at least one object from that heap. Their action can range from jremoving ust one to the entire heap. The players take turns, alternating until all of the objects are removed. The player who takes the last object from the final heap wins. For this report, I will use the example of a game of nim where there are two heaps with sizes 3 and 4 at the start. This start state would have the notation (3,4). Player A would move first, with Player B moving second.

Game Tree:

The root node is the initial state (3,4). The child nodes from this state can come from two outcomes. Either Player A removing objects from the first heap: removing 1,2,3 from the first heap gives (2,4),(1,4),(0,4) or Player A removes 1,2,3,4 from the second heap, giving us (3,3),(3,2),(3,1),(3,0). On Player B's turn, they can remove a nonzero amount from one of the nonzero heaps of the child state, producing many third-level child nodes. The tree continues until the goal state (0,0). Because the branching factor is the current heap sizes, the full tree is large even for two small heaps.

If you were to illustrate the tree, rather than describe it in text, it would show the root, which branches to 7 child nodes, which then each expand to several additional nodes. There are some game instances where the third-level child nodes could result in a goal state of (0,0).

Minimax Algorithm:

Starting from the root (3,4), Player A's move from (3,4) to (3,3) produces a Min node. At 2-ply we stop after Player B's next moves and evaluate those positions. By pulling from the first heap, we get the positions (2,3), (1,3), (0,3). From the second heap they would be (3,2), (3,1), (3,0). By using the algorithm, we see that every immediate reply from (3,3) still leaves Player A

with a winning (nim-sum ≠ 0) position, so (3,3) is a winning move in the 2-ply minimax evaluation.

Alpha-Beta Pruning:

Alpha-beta pruning works on the same minimax tree but keeps two values: α (the best Max score) and β (the best Min Score). It skips branches that can't possibly affect the final decision. For example, if we check the (3,3) child node first and find it gives a value of 1, α becomes 1. When evaluating other moves, if Min finds any reply with a value ≤ 1, the algorithm stops exploring that branch because Max already has a better option. Since most other moves lead to -1, alpha-beta quickly prunes out most of the tree. Testing the best move early (here it would be (3,3)) allows alpha-beta to ignore unnecessary calculations, as they may be identical to child nodes of that larger (3,3) node.

Analysis:

Minimax guarantees an optimal decision, but its cost is exponential (roughly O(b^d) where b is branching factor and d is depth). Alpha-beta pruning retains minimax's optimality guarantee while cutting the cost down dramatically. For Nim, the branching factor would be the size of each heap. Since the tree grows near-exponentially as you add heaps or increase heap sizes, naive minimax is not the best method. To conclude, minimax finds the correct move but is costly, while alpha-beta pruning can reduce that cost significantly while still being optimal.

Problem 2:

Optimization Techniques:

One common optimization technique to improve minimax performance is to use iterative deepening. We've discussed this in the class, but iterative deepening is when the algorithm searches shallow levels of the tree first, then gradually increases the depth to 2-ply, 3-ply, etc. while reusing information from earlier searches. This allows the algorithm to find a good move quickly even if there isn't enough time to search the whole tree. As I mentioned in response to the first problem, if you were to evaluate the node (3,3) like this, you would see that it has the most child nodes. (3,3)'s child nodes even include other nodes from the first group of child nodes. In a 2-ply Nim game, iterative deepening would look at immediate moves first, then expand to include the opponent's possible responses. This helps prioritize better moves early in the game and improves efficiency, especially when combined with alpha-beta pruning.

Heuristic Evaluation:

A heuristic evaluation function estimates how good a game state is without searching all the way to the end. This does only apply to nodes that are not the goal state of (0,0). For Nim, a simple heuristic could be based on the nim-sum: if the nim-sum is zero, that move leads to a losing position (with a score of -1), and if it's nonzero, it could lead to winning (with a score of 1). In a 2-ply game, instead of exploring all possible future states, the heuristic would assign these scores directly to child nodes, making the search much faster. While the heuristic-based minimax may not always find the perfect move in complex games, in Nim it performs almost as well as standard minimax because the nim-sum gives a strong mathematical indicator of which positions are good or bad.